Trustworthy Clients: Extending TNC for Integrity Checks in Web-Based Environments

---

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of Master of Science in Computer Science

in the University of Canterbury

by Sascha Rehbock

University of Canterbury

Department of Computer Science

and Software Engineering

2008

Contents

Abstract

Web-based services are vulnerable to a number of attacks. While providers of these services employ countermeasures (such as firewalls, encryption, and authentication systems) to reduce security risks, some of these security measures can be rendered useless if the PC of a user that accesses such a web-based service is not properly secured. Malicious software that is installed on a user's PC, for example, can potentially circumvent existing protection measures by recording login credentials and impersonating the victim.

To counter threats that are arising through client PCs, many providers of security sensitive web-based services have introduced usage policies for their services. These policies require users to ensure that their PCs are in a proper security state (e.g. the PC is equipped with an up-to-date anti-virus application, a personal firewall, and all security updates have been installed). However, service providers have no possible means of enforcing these policies and they have to rely on users to check the security state of their PCs manually.

This thesis presents a mechanism that allows a service provider to remotely measure the security state of a user's PC. This mechanism is based on Trusted Network Connect (TNC). TNC is a network access control mechanism that takes the security state of an access requesting party into account before making an access decision. However, TNC is currently limited to closed environments such as LANs and VPNs.

This thesis proposes solutions based on authentication standards for enabling TNC in open, web-based scenarios. In particular, an architectural model for TNC is proposed that takes additional security and privacy requirements into account. Furthermore, a communication scheme is proposed that is based on standardised protocols and message formats. These protocols and message formats have been leveraged to allow web-based TNC checks to be triggered through a Web browser and TNC messages to be exchanged.

These building blocks have been combined into a prototype implementation which has been evaluated using a test bed approach. This prototype successfully demonstrated that TNC can be adapted to web-based environments where it provides assurance as to the security state of clients accessing security sensitive web-based services.

Acknowledgements

I would like to thank my supervisor Associate Professor Ray Hunt for his invaluable support and an enjoyable co-operation. My thanks also go to Shane Balfe, Ingo Bente, and Mike Steinmetz for their feedback and comments during the early stages of this project. I also would like to thank my office colleagues Xianglin and Delio for the good company.

Special thanks go to my partner Daniela, for the countless hours of proofreading and for always being there for me. It is to her that I dedicate this work.

Finally, I would like to thank my family for their constant support and belief in me. *Danke.*

Chapter 1

Introduction

Today, using online services is an integral part of everyday life. Not only is the number of people that use the Internet increasing but also the scope of online services is broadening. It is now common to work, shop, and communicate online. The increasing popularity of online services, however, also attracts criminal and fraudulent activities, which range from attempts to make a PC unusable to stealing private or confidential information. Using online services is therefore also connected to some security risks. Some of these risks arise from threats that are caused by malicious software (malware), such as viruses, Trojan horses, or worms.

Online services that offer financial services, such as online banking, are a common target for attacks. Primarily, these attacks attempt to acquire sensitive information, such as log-in credentials. This information allows an attacker to impersonate a user and to initiate financial transaction using the victim's identity. Until recently, *phishing* attacks were amongst the most common form of attacks. The main goal of phishing attacks is to trick a user into entering access information into a web site that is under control of the attacker and which looks like a legitimate web site. Recently, however, attacks that are based on malware are becoming more common [Ant08]. In these attacks, users are tricked into installing software on their PC that allows an attacker to remotely capture keyboard input. The collected data is sent to the attacker and can then possibly be used to impersonate the user.

Furthermore, also online services that do not deal with financial matters are at risk. In some companies it is common to allow employees to access the company's Intranet remotely. While this allows the company's data to be accessed independently of one's location, it also introduces security risks. It is common practice to use authentication and encryption mechanisms for protecting access to an Intranet. Furthermore, most companies use firewalls and intrusion detection systems that make it difficult for an external attacker to gain access to the company's network. Attackers therefore might find it easier to get control of the PC of an employee and use it to piggyback into a company's Intranet, thus bypassing the encryption and authentication process [SJZD04]. After an intruder has gained access to the Intranet using this method, he or she is able to access and manipulate potentially business-critical data. Such an attack is difficult to detect from inside the company, as the attacker's

actions appear to be performed by the employee, and thus might not raise suspicion. This type of attack has recently become more common and an increasing number of companies are becoming victims of targeted malware attacks [Com07b].

Providers of security sensitive online services are aware of the threats that are related to malware. To mitigate the risks, service providers require users to accept usage policies before granting access to a service. These usage polices typically require a user to keep anti-virus software up-to-date, have a personal firewall running, and install the latest security patches for the operating system. Examples of these policies can be found when using online banking[1], dealing with government authorities[2], or in employment contracts that regulate remote access to a company's Intranet[3].

However, service providers have no possible means of actually enforcing these usage policies. They have to rely on the user to manually check the security state of their PC. Making sure that a PC is in a secure state, however, might overchallenge users. This can lead to situations in which users are overestimating the security state of their PC and accessing a service without having the required protection mechanism in place. What is more, other users might be afraid of not fulfilling the policy and therefore might avoid using a service with such a usage policy.

This thesis presents a mechanism that avoids these problems and can thus create confidence in the trustworthiness of a user's PC. Instead of relying on the user to check the security state of his or her PC, the mechanism presented here allows a provider of an online service to remotely measure the security state of a user's PC.

The idea of measuring a PC's security state has already been applied in other areas, such as network access control mechanisms. PCs and especially notebooks that are brought into a network might be carrying malware into a company's network, thus threatening the security of servers and other clients. Trusted Network Connect (TNC) is an emerging technology that tries to address this issue. TNC is essentially a network access control mechanism that takes the security state of a PC into account before allowing access to a network. This security state is called *integrity*. In TNC, a policy describes the integrity requirement. TNC software components that are executed on a client measure its integrity state and trans-

[1] The Code of Conduct published by the New Zealand Bankers' Association is used by all major banks throughout New Zealand. It states in its current version from July 2007 that online banking customers can be held liable for unauthorised transactions, if the customer is using a computer that "does not have appropriate protective software and operating system installed and up to date". The document can be retrieved from http://www.nzba.org.nz/pdfs/Code of Banking Practice 2007.pdf.

[2] It is, for example, necessary to accept a usage policy before the online services of Inland Revenue New Zealand can be used. The usage policy, as of 17.06.2008, states "We provide security to protect the website and Online Services. You are responsible for ensuring that your own computer is secure, including: taking all reasonable steps to prevent someone misusing or getting unauthorised access to your computer system or to our Online Services and ensuring your computer system and data are free of computer viruses."

[3] An employment contract of an international consultant company states: "The Employee will ensure that their computer has a high quality anti-virus programme and anti-spyware programme [installed] (such programmes being approved by [the company])".

fer the results to a TNC server which decides whether the client conforms to the integrity policy.

This thesis uses the principles of TNC integrity measurements and applies them to web-based applications. The result of this approach is a system that allows a service provider to assess a client's security state remotely. In addition to increasing the service provider's confidence in the trustworthiness of the client, it also increases the confidence of the user that his or her PC fulfils the requirements of the service provider.

1.1 Research Goals

The primary use case for TNC is access control in local area networks. Such an environment differs from online services in various respects. The overall question that this thesis strives to answer is therefore how TNC can be used in web-based environments. In order to answer this question it is necessary to analyse the differences between these environments with regards to access control. Networks in corporate environments can be considered as *closed* environments, as they are managed and controlled centrally by a single entity. By contrast, in web-based environments both *open* and *closed* scenarios exists. In an open scenario, entities are interacting with each other that are not centrally managed. It is therefore necessary to identify areas of TNC that need to be adapted so that it can be used in both open and closed web-based scenarios. Furthermore, protocol replacements need to be found that allow TNC communication in web-based environments. While adaptations are necessary to cope with the differences of the underlying environment, it is desirable to retain the main concepts of TNC. This allows the reuse of existing TNC principles and software. By using open standards to realise a web-based TNC check, the integration into existing authentication mechanisms is simplified. By implementing a prototype it can be demonstrated that TNC can be used in web-based environments.

1.2 Outline

This thesis is structured as follows. Chapter 2 gives an overview of TNC, its architecture, and related work. Chapter 3 analyses the different requirements of TNC in web-based environments and proposes changes to the architecture. These changes enhance the privacy of users and reduce complexity for service providers. Based on the proposed architecture, chapter 4 proposes a mechanism for encapsulating the result of a TNC check using standardised methods and describes the message exchange in a web-based TNC check. In TNC, integrity measurements requests and integrity measurements are exchanged between a client and the party that makes the access decision. These messages need to be encapsulated so that they can be transported in a web-based environment. A transport mechanism

for these messages based on open standards is developed in chapter 5. A mechanism is necessary that allows triggering of TNC checks in web-based environments, that is, through a browser. Such a mechanism is proposed in chapter 6. Based on the proposed architecture, protocols, and mechanisms a prototype has been implemented. This implementation and the results of performance tests are the subject of chapter 7. Finally, chapter 8 summarises this thesis and reviews possible future work.

Chapter 2

Background and Related Work

The idea of determining a PC's security state as part of an access control mechanism is already applied in local area networks. Traditional Network Access Control (NAC) systems focus mainly on authentication mechanisms, that is, on preventing unauthorised devices connecting to a local area network. However, they fail to prevent network devices with a questionable state of security from connecting to the network. TNC closes this gap and takes the client's integrity status into account. This chapter gives an overview of TNC and related technologies. The architecture of TNC is introduced in section 2.1. Before an access decision for a client can be made, its integrity state must be assessed and compared to a policy. Section 2.2 briefly describes this process. TNC can use the infrastructure of an existing network access control system, as section 2.3 summarises. The functionality of TNC can be extended by using Trusted Platform Modules, as summarised in section 2.4. Finally, section 2.5 gives an overview of related work.

2.1 Trusted Network Connect - Architecture Overview

The aim of TNC [Tru07b, Tru07e, Tru07g, Tru07c, Tru07d, Tru07f] is to base a network access decision on the security state of a network endpoint. In the context of TNC, this security state is called *integrity*. TNC defines integrity as the "relative purity from software that is considered harmful". The presence of anti-malware software or personal firewalls can protect an endpoint. Their presence can thus be used as an indicator that an endpoint is not infected with malicious software [Tru07b].

The architecture of TNC has been designed openly in order to be vendor and technology neutral. The architecture, as specified in [Tru07b], is depicted in figure 2.1[1] and described in the following.

The TNC architecture is composed of three horizontal layers and three entities. These entities, illustrated as columns, are: the Access Requestor (AR), the Policy Enforcement Point

[1] Based on figure 2 [Tru07b] on page 13.

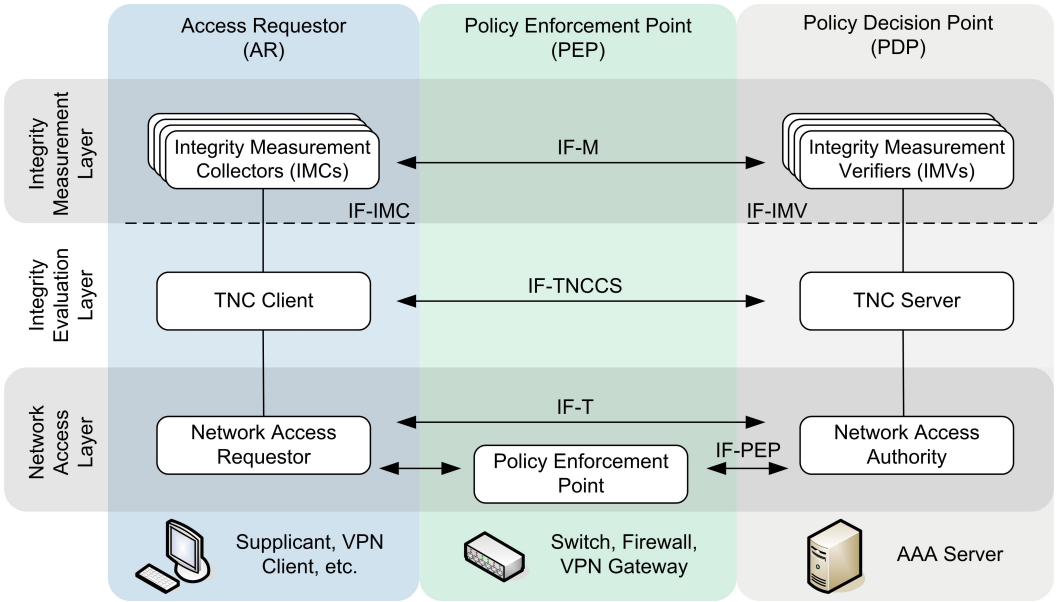


Figure 2.1: Overview of the TNC architecture

(PEP), and the Policy Decision Point (PDP). Each entity consists of three components. Components are grouped into the same layer based on a similar function or role they fulfil inside the architecture. They are communicating with each other using interfaces defined by the TNC specifications. These interfaces are depicted as named lines and arrows. In the following, details of the concept of entities, components and layers are given.

2.1.1 Entities in the TNC Architecture

The TNC architecture describes three logical entities that interact with each other. The actual physical form of an entity is not dictated by the architecture. An entity might be implemented as a software component or as a set of redundant machines depending on its functionality and the deployment's needs. The entities in TNC's architecture have to fulfil the following roles:

Access Requestor (AR) The AR is a client, for example a PC or a notebook, seeking access to a network. Its integrity state is measured and reported to the Policy Decision Point.

Policy Decision Point (PDP) The PDP is the entity which makes the decision about whether the AR should be granted access to the network. These decisions are based on policies which describe the integrity requirements a client must fulfil.

Policy Enforcement Point (PEP) The PEP is the entity, for example a switch, that is responsible for enforcing the decision made by the PDP.

Each of the entities described above can be further divided into components that interact with each other in different layers.

2.1.2 Layers in the TNC Architecture

The TNC architecture defines three horizontal layers which are described in the following.

Network Access Layer (NAL) At this layer, the physical communication on the network takes place. All three entities have components that are operating in this layer. Technologies that are used in this layer include IEEE 802.1X[2] and Virtual Private Network (VPN)[3].

Integrity Evaluation Layer (IEL) Components in this layer collect integrity measurements from the Integrity Measurement Layer. Based on these measurements and a combining policy, components in this layer are responsible for evaluating the overall integrity of an Access Requestor.

Integrity Measurement Layer (IML) At this layer, plug-in components are responsible for collecting and verifying integrity-related information. These components consist of two parts. The client-side part is responsible for collecting integrity information in a certain domain, such as, for example, a specific anti-virus product.

On the server side, that is, the Policy Decision Point, the collected information is verified and compared to an integrity policy. The result of this comparison, that is, a recommendation about whether the Access Requestor should gain access to the network, is then passed to the Integrity Evaluation Layer.

2.1.3 Components in the TNC Architecture

As mentioned before, entities in the TNC model can be divided into components that act at different layers. These components and their responsibilities are described briefly in the following.

2.1.3.1 Access Requestor

The following components are part of an Access Requestor:

Network Access Requestor (NAR) The NAR is a software component that is responsible for establishing network access. After a PC is connected to a network, it controls all steps that are necessary to gain access to a network (e.g. authentication and TNC integrity check).

TNC Client (TNCC) The TNCC is a software component running on an AR. Its main responsibility is to aggregate information gathered from different Integrity Measurement

[2] See section 2.3.1.

[3] A VPN allows the creation of a private communication tunnel in a public network. More information about VPNs can be found, for example, in RFC4364 (http://tools.ietf.org/rfc/rfc4364.txt).

Collectors (IMCs, see below) and to send this information to the PDP. In addition, the TNCC is also responsible for starting all IMCs that are available on a system.

Integrity Measurement Collector (IMC) IMCs are responsible for measuring security aspects of the AR's integrity. Examples for security aspects include anti-virus software parameters, personal firewall status, or the existence of security patches. IMCs are implemented in software and have a counterpart at the PDP, which is called Integrity Measurement Verifier (IMV). An IMC (and the related IMV) is responsible for only one particular aspect (e.g., only for a certain anti-virus software product). It is therefore possible to install multiple IMCs on one Access Requestor. According to the TNC specification, it is expected that IMCs and IMVs are part of future releases of security related software applications [Tru07c].

2.1.3.2 Policy Enforcement Point

Policy Enforcement Point (PEP) The Policy Enforcement Point, which is the only component that exists in a PEP entity, controls access to the network. This access control mechanism is based on the access decision made by the PDP. An examples for a PEP is a network switch.

2.1.3.3 Policy Decision Point

Network Access Authority (NAA) The NAA is the component in the TNC architecture that decides whether a client, that is, an Access Requestor, should be granted access. This decision is based on a recommendation given by the TNC Server. The NAA is typically implemented as part of an AAA (Authentication, Authorisation, and Accounting) Server, for example a RADIUS server[4].

TNC Server (TNCS) The TNCS manages the communication between IMCs and IMVs. Action recommendations are gathered from these components and, based on a policy, combined into one access recommendation which is then passed to the NAA.

Integrity Measurement Verifier (IMV) IMVs are the server-side counterpart of IMCs. They collect and analyse integrity measurements for a certain security aspect received from a corresponding IMV. The received measurements are then compared to an integrity policy and an access recommendation (for one particular aspect) is given to the TNCS.

[4]See section 2.3.3 for a description of RADIUS.

2.2 Integrity Measurement Message Exchange

In the previous section, an overview of the TNC architecture and the responsibilities of its components was given. This section gives an overview of the interaction between components and the related message exchange. Figure 2.2[5] shows which steps are performed during a TNC check. These steps are described in the following [Tru07b].

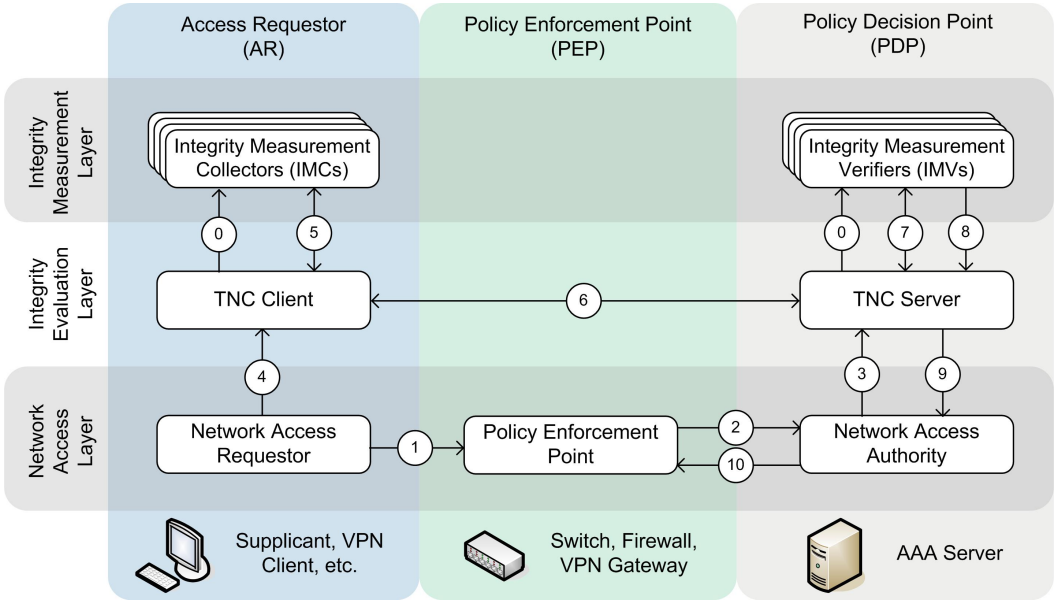


Figure 2.2: Overview of the message flow during a TNC integrity check

Step 0 Before an integrity check can be performed, Integrity Measurement Collectors are loaded and initialised by the TNC Client on the Access Requestor. Similarly, all available Integrity Measurement Verifiers on the Policy Decision Point are initiated by the TNC Server. As part of this initialisation process, policies are loaded that state the integrity requirements for accessing the network.

Step 1 An integrity check is always triggered by the Access Requestor. If a connection to a network shall be established, the Network Access Requestor component sends a *connection request* to the Policy Enforcement Point. This step can either be initiated manually by a user, or automatically, for example, after detecting that a physical network connection was established.

Step 2 The Policy Enforcement Point sends a *network access decision request* to the Network Access Authority, which indicates that an Access Requestor is seeking access to the network. This step typically triggers an authentication process with the user of the Access Requestor. If this authentication fails, the TNC integrity check is aborted.

[5]This figure is simplified in favour of clarity and does not contain optional steps, such as platform authentication.

Step 3 The Network Access Authority informs the TNC Server about the access attempt and requests a TNC access recommendation.

Step 4 In this step, the TNC Client is informed that an integrity check shall be performed.

Step 5 The TNC Client informs all installed Integrity Measurement Collectors that a new connection request has been made and that an integrity check needs to be performed. Following its initiation, an Integrity Measurement Collector collects aspects of the security state of the Access Requestor and reports them back to the TNC Client. The interface used between TNC Client and Integrity Measurement Collector is called IF-IMC. It is defined in [Tru07c].

An example for an integrity measurement result can be found in listing 2.1[6]. This example shows the result of an Integrity Measurement Collector for the anti-virus software *VirusScannerPro* created by the vendor *AcmeAntiVirus*. It contains the version of the installed software and the date it was last updated.

```
<AcmeAntiVirusIMC version="1.0">
  <VirusScannerPro installed="true" running="true">
    <version>5.2</version>
    <lastUpdate>2007-08-15</lastUpdate>
  </VirusScannerPro>
</AcmeAntiVirusIMC>
```

Listing 2.1: Example of an integrity measurement result collected by a fictive IMC

The format for expressing and exchanging integrity measurements between Integrity Measurement Collectors and Integrity Measurement Verifiers is not specified by TNC. This interface is referred to as IF-M by the TNC specification and producers of integrity measurement components can define their own protocols for this interface.

Step 6 The TNC Client collects the results from all Integrity Measurement Collectors and combines them into one message. This message is transferred to the TNC Server. It is worth pointing out that, although the communication takes place on the logical Integrity Evaluation Layer, these messages are physically transferred via the Policy Decision Point[7].

While the format for expressing integrity measurements (IF-M) is not specified, the TNC specification defines the interface between TNC Client and TNC Server. This interface is called IF-TNCCS [Tru07g] and defines a message format for exchanging information between TNC Client and TNC Server. A message between these parties is transferred in a *TNCCS-Batch*. Listing 2.2 shows an example of such a TNCCS-Batch, which originates from a TNC Client and is sent to a TNC Server. It contains

[6] Based on [Wut06, page 70].
[7] As described in section 2.3.

two messages. The first message is a control message (sent from TNC Client to TNC Server) that indicates the language in which messages that are shown to users are expected (lines 3–8). The second message contains an integrity measurement result that is sent from an Integrity Measurement Verifier to an Integrity Measurement Collector (lines 9–12). Each Integrity Measurement Verifier and Collector is associated with an identifier[8]. This identifier (line 10) allows for the routing of messages to the correct Integrity Measurement Verifier or Integrity Measurement Collector respectively. An integrity measurement can either be encoded in Base64[9] (line 11) or in XML form (as shown in listing 2.1.)

```
1  <?xml version="1.0" ?>
2  <TNCCS-Batch BatchId="1" Recipient="TNCS">
3   <TNCC-TNCS-Message>
4    <Type>00000003</Type>
5     <XML>
6      <TNCCS-PreferredLanguage>en</TNCCS-PreferredLanguage>
7     </XML>
8   </TNCC-TNCS-Message>
9   <IMC-IMV-Message>
10   <Type>00299802</Type>
11   <Base64>V2luZG93c3w1fDF8MjYwMHwyfFNlcnZpY2UgUGFjayAzfA==</Base64>
12  </IMC-IMV-Message>
13 </TNCCS-Batch>
```

Listing 2.2: Example of a TNCCS-Batch

Step 7 The TNC Server receives the TNCCS-Batch from the TNC Client and extracts the included messages. Each measurement is then transferred to the specified Integrity Measurement Verifier. Each Integrity Measurement Verifier compares the received measurement with the policy that has been loaded during the component's initialisation phase. Listing 2.3 shows an example policy file[10]. This policy is specific to an Integrity Measurement Verifier of an anti-virus application. It requires that the software version is 5.0 or greater and that the last update is not older than three days.

```
<?xml version="1.0"?>
<Policy>
  <All>
    <version compare="greaterThan">5.0</version>
    <lastUpdate compare="notOlderThan" type="days">3</lastUpdate>
  </All>
</Policy>
```

Listing 2.3: Example policy for a fictive IMV

[8] Cf. [Tru07g, section 2.8.4] for details about the composition of this identifier.

[9] An arbitrary string that is Base64 encoded can be expressed using only numbers, upper- and lower-case letter, and the "=" character. See RFC4648 for more information (http://tools.ietf.org/rfc/rfc4648.txt).

[10] Based on [Wut06, page 94].

If the Integrity Measurement Verifier is not able to decide whether the policy requirements have been met, it can request more information from the Integrity Measurement Collector. Steps five, six, and seven are then repeated to retrieve additional information.

Step 8 In this step, each Integrity Measurement Verifier sends an access recommendation to the TNC Server. The interface between these components is called IF-IMV and is defined in [Tru07d]. This access recommendation only refers to the specific security aspect that is dealt with by an Integrity Measurement Verifier. Such a recommendation consists of two statements: an *Evaluation Result* and an *Action Recommendation*. The first statement indicates whether the Access Requestor complies with the policy. If the AR is non-compliant with the policy, the IMV can indicate if the deviation is minor or major. The Action Recommendation indicates whether the Access Requestor should be granted access to the network based on the Integrity Measurement Verifier's policy [Tru07d, section 3.5.7 et seq.].

Step 9 Based on the recommendations of all Integrity Measurement Verifiers, the TNC Server - based on a combining policy - derives its final access recommendation, which is then passed on to the NAA.

Step 10 The Network Access Authority receives the recommendation from the TNC Server. However, as it is only a recommendation, the decision to grant or deny access lies with the Network Access Authority. The final decision is sent to the Policy Enforcement Point where it is enforced. In addition, the Network Access Authority also sends its decision to the TNC Server, which in turn informs the TNC Client about the outcome of the integrity check.

The steps described above are performed during the *assessment* phase of a TNC check. In addition to this phase, the TNC architecture defines two additional phases, that is, *isolation* and *remediation*.

If an Access Requestor does not meet the integrity requirements, it can be isolated to protect other entities in the network from potential malware that is installed on this Access Requestor. In that case, the Policy Enforcement Point is instructed by the Policy Decision Point to restrict access to a certain isolated part of the network.

After an Access Requestor has been isolated, the remediation phase can commence. In this phase, Integrity Measurement Collector components receive integrity-related updates from their corresponding Integrity Measurement Verifiers. After these updates have been installed successfully, the assessment phase is re-initiated and the Access Requestor can be granted access to the network [Tru07d].

As already mentioned earlier, TNC was designed to allow multiple underlying technologies to be used to transport TNCCS-Batches. One such technology is standardized as 802.1X. Its use in conjunction with TNC is described in the next section.

2.3 Network Access Control with TNC and IEEE 802.1X

The TNC architecture was designed to allow interoperability with existing standards. For network access control, a combination of 802.1X, EAPoL, and RADIUS is commonly deployed [EA03, chapter 8]. TNC can be used on top of the aforementioned technologies. This section gives a brief overview of these technologies and shows how they can be integrated with TNC.

2.3.1 IEEE 802.1X

Networks without access control are accessible by every device that can be physically connected to it. However, unauthorised devices connected to an organisation's network pose a threat to network security. The IEEE standard 802.1X [LAN04] addresses this issue by defining a port-based network access control mechanism for the IEEE 802 network family[11]. In the following only *wired* 802.3 Ethernet is considered [LAN04].

In this context, port-based means that access control starts at the point where a client is connected to a network, for example by connecting a PC to a port of a network switch. The network switch blocks all network traffic at this port until the device (such as a PC) is authenticated successfully. The following three components exist in an 802.1X infrastructure:

- The client trying to access a network is called a *Supplicant*. This role corresponds with the Access Requestor in TNC.
- The *Authenticator*, for example a network switch, blocks all traffic from Supplicants that have not been successfully authenticated. This role is identical to the Policy Enforcement Point in TNC.
- The *Authentication Server* authenticates the Supplicant. In addition, it instructs the Authenticator to allow a Supplicant to access the network, that is, opening the port to which the Supplicant is connected to, if the authentication was successful. In the TNC model, this role is performed by the *Policy Decision Point*.

Figure 2.3[12] shows the message exchange during an authentication in an 802.1X environment.

After the Supplicant has been (physically) connected to the network, the Authenticator queries the identity of the Supplicant. This identity, for example in the form of a user name, is relayed by the Authenticator and sent to the Authentication Server. In the following authentication process, a Supplicant must prove its identity, for example by proving the knowledge of the correct password.

[11] For example Ethernet (IEEE 802.3) and Wireless LAN (IEEE 802.11).
[12] Based on figures 8-2 and 8-3 in [LAN04].

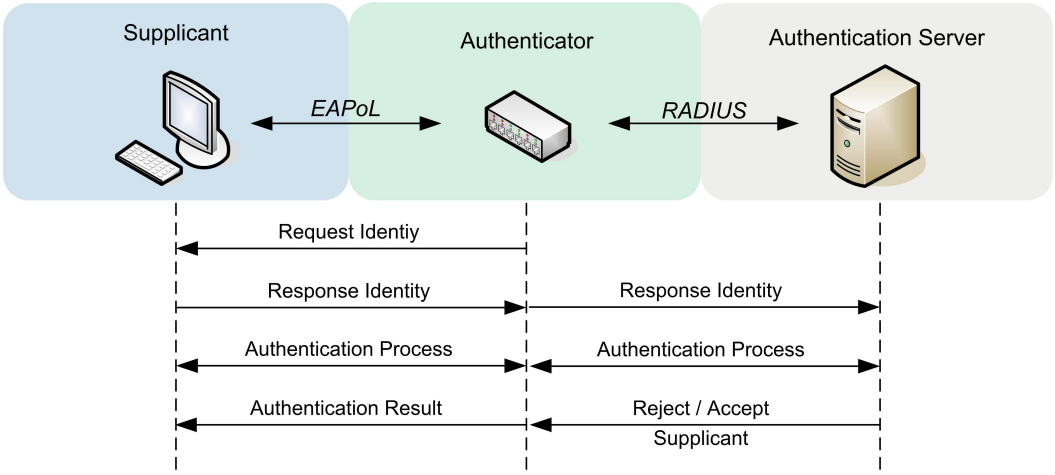


Figure 2.3: Requesting access in IEEE 802.1X using EAPoL and RADIUS

If a Supplicant successfully proves its identity (and is allowed to access the network), the Authentication Server instructs the Authenticator to grant access. The Authenticator will then inform the Supplicant about the Authentication Server's access decision.

The message exchange between Supplicant and Authenticator is based on EAPoL [LAN04, page 23]. The communication between Authenticator and Authentication Server is based on RADIUS[13]. The Authenticator is responsible for converting EAPoL messages to RADIUS messages and vice versa. However, it does not need to understand the messages' content.

The two protocols used in 802.1X, that is, EAPoL and RADIUS, are briefly introduced in the following sections.

2.3.2 EAPoL

EAPoL (EAP over LAN) is defined as part of the 802.1X specification. It defines a container protocol in which EAP messages can be transported in Ethernet based networks for enabling Ethernet-based authentication.

EAP (Extensible Authentication Protocol), which has emerged from dial-in networks [EA03, pages 120 et seqq.] is defined in RFC 3748 ([Abo04]). It is a framework-like protocol for client authentication that can be used for different authentication methods. Existing methods include simple user name/password based approaches and mechanisms that use Transport Layer Security (TLS) mechanisms for performing an authentication[14]. However, it is possible to define new methods and use them within the EAP framework. TNC uses this mechanism to integrate into existing EAPoL environments[15].

[13] While this protocol choice is not mandatory, it is proposed in the 802.1X specification [LAN04, page 37].
[14] Section 5.2.1 describes such mechanisms.
[15] See section 2.3.4.

2.3.3 RADIUS

RADIUS (Remote Authentication Dial In User Service) is defined in RFC 2865 [Rig00] and is based on UDP[16]. Its purpose is to transport messages related to network authentication, authorisation, and accounting in an IP based network. In 802.1X, RADIUS is used for the communication between Authenticator and Authentication Server. That is, it is used for transferring authentication messages (between Authenticator and Authentication Server) and for expressing access control decisions to the Authenticator. Additional attributes in access decision messages, as defined in [Zor00], allow for these decisions to be flexible and fine grained. Furthermore, RADIUS can be used for transporting EAP messages[17].

2.3.4 Using TNC with 802.1X

In section 2.2, an overview of the message exchange during a TNC integrity check was given. TNCCS-Batches are used to exchange measurements and measurement requests between TNC Client and TNC Server. For sending these XML-based messages, network protocols are required that work on the lowest layer of the TNC architecture, that is, the Network Access Layer[18]. The TNC specification describes how the protocols used in IEEE 802.1X can be applied at this layer.

Similar to IEEE 802.1X, two communication channels with separate interfaces exist in the TNC architecture[19]. The interface for exchanging TNCCS-Batches on the network layer is called IF-T [Tru07f]. In addition, the interface IF-PEP defines the communication between Policy Decision Point and Policy Enforcement Point [Tru07e].

By applying 802.1X to IF-T, it is subdivided into two separate communication channels. The first channel transfers messages between Access Requestor and Policy Enforcement Point. According to 802.1X, this communication channel is realised using EAPoL. TNC defines a new EAP authentication method[20]. Instead of exchanging, for example, user name and password information, the newly defined method *TNC-EAP* is used to transfer TNCCS-Batches.

The second communication channel between Policy Enforcement Point and Policy Decision Point is based on RADIUS. The Policy Enforcement Point acts as a *protocol translator* [Tru07f, section 3] and translates incoming EAPoL messages to RADIUS messages. Because RADIUS can be used to encapsulate EAP messages, the EAP authentication method TNC-EAP can be reused. Figure 2.4 shows the EAP and RADIUS messages that are exchanged during a TNC integrity check.

[16] See for example [Tan03, chapter 6.4] for more information about the User Data Protocol.
[17] This extension is defined in RFC 3579 [AC00].
[18] See section 2.1.2.
[19] Cf. figure 2.1 on page 16.
[20] Cf. section 2.3.2.

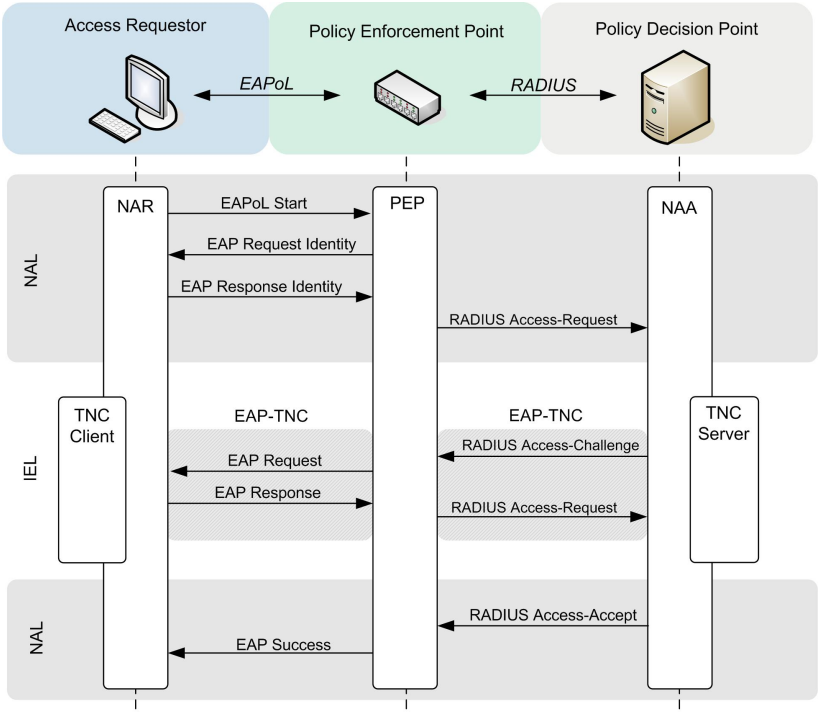


Figure 2.4: Message exchange on the Network Access Layer (NAL)

When an Access Requestor is connected to a network, for example by physically connecting it to a network port, an 802.1X-based communication is established as described in section 2.3.1.

Following the initial authentication, the TNC integrity check begins. The TNC Client creates TNCCS-Batches that are sent to the Policy Enforcement Point via TNC-EAP (using *EAP Requests* and *Responses*). The Policy Enforcement Point forwards these messages encapsulated in RADIUS *Access-Challenge* and *Access-Request* messages.

After all integrity checks have been performed and an access decision has been made, the Policy Decision Point sends the access decision the Policy Enforcement Point for enforcement. Using IF-PEP, this decision is encapsulated in a RADIUS *Access Accept* or *Reject* message. By making use of RADIUS attributes, three different models of access control can be expressed. These models are called isolation levels:

Binary-based isolation is the simplest form of isolation. Access to the network is either allowed or completely denied.

VLAN-based isolation can be used to isolate Access Requestors in a dedicated network area, in which they can obtain security updates. These areas are called redemption areas[21].

[21] RADIUS tunnel attributes, defined in RFC 2868 [Zor00], are used to define which VLAN an Access Requestor shall be assigned to. The simplest approach using VLAN-based isolation uses two VLANs: one for Access Requestors that comply with the policy and one for those that do not comply.

Filter-based isolation allows for the definition of filter rules that are consulted for enforcing an access decision. RADIUS defines a *Filter-Id* attribute that is used to control which filter should be applied to a certain port, that is, for a certain Access Requestor. Using filters, it is possible to grant, limit, or deny access to the network.

After the Policy Enforcement Point has enforced the access decision, the Access Requestor is informed about it using an EAP message.

2.4 TNC and Trusted Platform Modules

Similar to TNC, the concept of *Trusted Platform Modules (TPMs)* has been developed by the Trusted Computing Group (TCG) [Tru07a]. TPMs are hardware components that are part of a PC's mainboard[22]. A TPM provides cryptographic functions, such as hashing and signing. It is further tamper-evident, meaning that manipulations can be detected and it cannot be removed from its platform (e.g., a PC). Because of this robustness, it is referred to as the *root-of-trust* and it is assumed to be trustworthy. The TNC specification describes how TNC can benefit from TPMs [Tru07b, Tru07f].

Firstly, TPMs have the ability to store information in a protected storage. Because TPMs are implemented in hardware, this storage is protected against attacks that originate from software, such as malware. This storage is therefore ideal for storing sensitive information, such as cryptographic (private) keys. TNC can make use of keys that are stored in a TPM for performing a *platform authentication*. During such an authentication, it is not the user that has to prove his or her identity, but the PC that is used to access a network.

Secondly, TPMs can perform integrity measurements. Unlike TNC integrity checks, TPM integrity checks are based on the cryptographic digest (or hash) of a (hard- or software) component. For example, a program's integrity measurement can be calculated using the hash of its instruction sequence, that is, the executable file itself and its input [GM06]. By combining the hashes of all programs and their configuration that are executed on a platform, the platform's configuration can be expressed using a single hash value. TNC can make use of this mechanism for ensuring that only genuine Integrity Measurements Collectors are loaded by comparing their expected hash value with the hash value obtained using TPM support. In addition, Integrity Measurements Collectors can report integrity measurements that have been gathered using a TPM to Integrity Measurements Verifiers. Because TPMs work on a hardware level, these measurements cannot be forged, for example, by malicious software that tries to hide its presence.

The TNC specification gives an architectural overview that shows how TNC can be linked to TPMs. The interface between TNC and TPMs is called IF-PTS (*Interface - Platform Trust*

[22] In addition, TPMs are also used in other devices, such as mobile phones. See https://www.trustedcomputinggroup.org/groups/mobile for more information.

Service). While the specification states general requirements for IF-PTS [Tru07b, section 6.4], no details have been published that describe how IF-PTS can be implemented. Before TNC can benefit from TPMs, it is therefore necessary that the TCG publishes these details.

While specification about linking TPMs and TNC have not yet been published, it is possible to use TPM functionality without relying on TNC. In the following section, other approaches for performing integrity checks are briefly summarised.

2.5 Related Work

TNC combines integrity measurements and network access control. In addition to TNC, further approaches have been proposed that address similar areas. Such an approach is *Remote Attestation*, which has been developed by the TCG [Tru07a]. As described in the previous section, TPMs can be used to assess the integrity of a system. Using remote attestation, this information can be made available for a challenger over a network.

In [SJZD04] a system is proposed that uses remote attestation for enforcing integrity policies when accessing Intranets using a VPN protected communication channel. The presented approach is similar to TNC, as a remote client must prove its integrity state before access to the network is granted. However, it lacks mechanisms such as remediation, which are part of TNC.

Other work, such as [STRE06] and [CHJ07] concentrates on how remote attestation can be used reliably and provide solutions to mitigate masquerading attacks. Similarly, [GPS06] proposes to bind the integrity measurements gathered through a TPM to an SSL channel that is used for communicating with the remote challenger.

[YEN+05] presents a system that exchanges remote attestation messages using a SOAP-based Web service. Furthermore, [NVH07] focusses on policy-based access control that uses remote attestation for accessing Web services.

A general drawback of remote attestation is that it is based on hash values that have been determined by a TPM. In order to check which software is executed on a platform, databases of programmes and their related hash values must exist. As the hash of a programme changes with every update, these databases will grow steadily. In addition, software often relies on external libraries which also have to be considered in these databases. Furthermore, behaviour of software often depends on its configuration. Storing and managing hash values for all possible configurations of complex software, such as an operating system, can become impractical [Hal06].

A property-based attestation was proposed in [SS04], which addresses some shortcomings of remote attestation. Instead of relying on a specific hardware or software configuration, this mechanism focusses on the properties that a platform offers. This requires that a state

of a platform is mapped to a property. These properties are then asserted by a third party [SS04]. [CLL+06] proposes a protocol for property-based attestation, while [KSS07] presents an implementation thereof. However, unlike TNC, no mechanisms for access control have been integrated into property-based attestation.

While TNC has been designed for network access control, previous work proposes to use it in different environments. [SLM06] proposes the use of TNC to protect access to Web services, while [BM07] suggests the idea of applying TNC to prevent cheating in online games. [SLM06] and [BM07] describe how the entities of the TNC architecture could be reused in other environments. However, both approaches only touch the surface while describing how TNC can be used in environments other than networks. They fail to address how the existing EAP/RADIUS-based communication can be applied for Web services or online gaming. Furthermore, they do not take into account that applying TNC in other environments can affect a user's privacy.

Previous work on new applications for TNC applies the TNC architecture without adapting it to the specific properties of the new environment. By contrast, this thesis adapts TNC for its use in web-applications where it is necessary. Furthermore, this thesis also presents the first implementation of TNC outside the scope that is covered in its specification. The following chapter proposes changes to TNC's architecture in order to adapt it to web-based environments.

Chapter 3

Adapting the TNC Architecture

The previous chapter outlined aspects of the TNC specification. Part of this specification is the TNC communication model. It specifies how information is exchanged during a TNC integrity check and is currently used in local area network use cases. When compared to a web-based scenario, two differences are apparent. Firstly, in web-based environments no centralised Policy Enforcement Point (PEP, e.g. a switch) exists that centrally handles all access requests. Secondly, instead of accessing one entity (i.e., one network), a user is likely to access multiple services (such as online banking, e-commerce sites, and a company's Intranet). These differences result in different requirements, as described in the following section.

3.1 Requirements

In TNC scenarios described in the TNC specification, a user seeks access to a single service, that is, a network. Before access to this service is granted, the user's PC is isolated from the rest of the network and no other services can be accessed, that is, the PC cannot communicate with any party other than those that are participating in the TNC integrity check (PEP and PDP). In a web-based environment, however, other services are accessible for a user before, while, and after a TNC check is performed. This allows for the consideration of more flexible architectural approaches. For example, PEP and PDP can be detached and do not need to be managed by the same organisation. However, because a user is exposed to other parties during a TNC check, additional security and privacy requirements exist.

As mentioned above, in the original TNC model one centralised Policy Enforcement Point exists that handles all access requests. Because a user has the choice of several services that can be accessed in the web-based scenario, more than one PEP is required. Figures 3.1 and 3.2 summarises this relation.

Because of the aforementioned differences, several requirements exist in a web-based scenario that need to be considered when designing a web-based TNC integrity check. These requirements are discussed below.

Figure 3.1: TNC in a LAN environment: A single PEP controls access of an Access Requestor (AR) to a single service (i.e., the network).

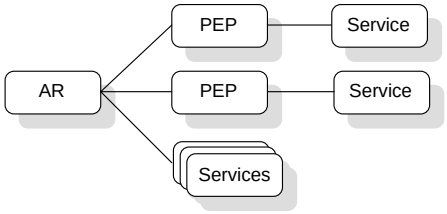


Figure 3.2: TNC in a Web environment: Several services and PEPs can be accessed.

Trustworthy Verifier A TNC integrity check reveals detailed information about which software is installed and executed on a PC. A malicious verifier could use this information to perform an attack targeting vulnerabilities of these software products. Knowing, for instance, that a system is not issued with the latest operating system patches gives an attacker opportunities and hints about how to attack [YEN+05]. In a local (corporate) network this threat scenario is relaxed as the network is centrally managed and only a single authentication system performs the integrity check. By contrast, in a web-based scenario, potentially any service offering party (including malicious ones) can request an integrity check. It is therefore important to protect the user from performing an integrity check with a malicious verifier. Furthermore, it is required that integrity measurements are only sent in encrypted form to prevent an attacker from eavesdropping.

Privacy Assurance As stated above, detailed information about a PC is revealed during an integrity check. Users may have privacy concerns when it comes to releasing this information. The situation is less problematic in a corporate environment, where this information is contained within a company's perimeter and can be centrally managed. Users can expect that information gathered during the TNC integrity check is only used to determine the compliance with the security policy and is not disclosed.

When performing a TNC check in an uncontrolled network, such as the Internet, however, the integrity measurements are at a greater risk of being used inappropriately. A user is interacting with several, potentially unknown parties. It is therefore important to integrate privacy protecting mechanisms into the web-based TNC check, which limits disclosure of integrity measurements to only those parties that have a genuine need to know them.

Usability and User Experience Unlike in a corporate environment, IT service staff are not or only to a limited extent available for users of Internet-based services. It is therefore essential to keep the process of TNC integrity checks simple and transparent for the user.

The original TNC architecture has to be adapted to fulfil the above requirements. While it might not be possible to completely satisfy every requirement, workable compromises can be developed. Before introducing different models, the next section introduces the basic entities that can be found in each of them.

3.2 Entities and their Roles

The TNC architecture and its entities are tailored to a network use case. Before the existing model can be adapted to web-based environments, it is helpful to define entities which reflect roles and responsibilities when performing a TNC check in a web-based environment. In particular, these entities focus on the concept of using a service rather than accessing a network. In the following, three entities are introduced which are mapped to the entities in the original TNC architecture.

Client A client is an entity seeking access to an application. *Client* in this context does not mean a user (i.e., a person) but rather his or her PC and the installed software stack that is used to access an application. Furthermore, a client is also the target of an integrity check. The client corresponds to the *Access Requestor* in the TNC model and contains the following three TNC components: Integrity Measurements Collectors, TNC Client, and Network Access Requestor.

Service Provider A service provider (SP) offers a (security sensitive) application or service which is accessed by a client. Before access to a particular application is granted, an SP requires assurance about the security state of the client. The SP specifies its integrity requirements in a policy statement. For the actual measurement, the SP relies on an entity called Verification Service Provider (see below). In addition to offering a service, an SP fulfils the roles of the Network Access Authority and the Policy Enforcement Point, as defined by the TNC specification.

Verification Service Provider The Verification Service Provider (VSP) performs the actual integrity check and reports the result to the SP. In order to fulfil this role, a VSP performs the tasks of Integrity Measurement Verifiers and the TNC Server.

These entities are the base for web-based TNC models that are discussed in the following section.

3.3 Attestation Models

By using the entities described in the previous section, different models can be constructed. These models and their properties are discussed in the following.

3.3.1 Direct Attestation

The *direct attestation* model involves two parties: A client and the SP, which also acts as the VSP. That is, it is offering a service and is also performing TNC integrity checks. While this is the simplest model, it also requires the highest level of functionality to be provided by the SP. This model, which is an adaptation of the server-based PEP model [Tru07e, section 9.1.3], is depicted in figure 3.3, .

After a TNC enabled client has accessed a Web application (1), the SP can send a TNC integrity check request (2). Upon approval by the user, the client TNC software responds by starting the integrity check (3 and 4). After the TNC measurements have been sent, a decision about the policy compliance of the client can be made. The client is informed about the decision in the last step (5).

From a user's point of view, this model appears to be trustworthy as users are interacting with a trustworthy and known party (e.g. a bank, or their company). However, this model may raise privacy concerns as all integrity information is sent to the SP. In general, an SP does not need to know exactly which software is running on a client's machine, that is, the exact integrity measurements. It is sufficient for the SP to know whether the integrity state of a user's machine complies with the integrity requirements of the SP. An approach is therefore desirable that is more protective with regards to privacy.

As a large number of vulnerabilities are discovered every day [Com07a], software vendors release updates for their security software frequently. Making educated decisions about the security state of a user's PC makes it necessary to keep track of different versions of security related software and their updates. It is likely that offering such services is not within the main business scope of an SP and hence, the direct attestation approach is not very practical. It is more likely that this task is performed by a separate third party which solely performs the role of the Verification Service Provider. This approach is considered in the following section.

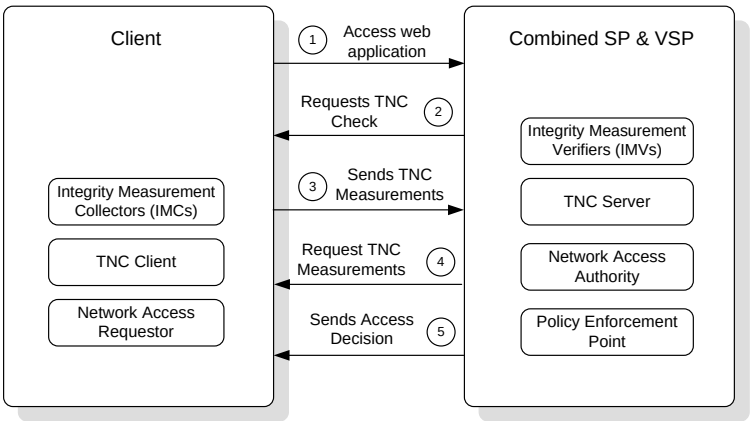


Figure 3.3: Message exchange in the direct attestation model

3.3.2 Relayed Attestation

In the *relayed attestation* model the Verification Service Provider (VSP) acts as an independent party. The VSP can offer its verification services to a number of SPs. Figure 3.4 depicts this approach.

As it is possible that each SP has different integrity requirements, SPs have to express their requirements in a policy that is sent to the VSP (0). As described in the previous section, an SP is not keeping track of security software and their updates and is therefore unable to state detailed policy instructions. Instead of low level measurement directives and expected results, the policy therefore contains high level security goals (e.g. virus protection up-to-date, operating system issued with latest patches, and personal firewall is installed and running). A mechanism on the VSP translates the high level policy into a low level policy that can be matched against the integrity measurement results.

From a client's perspective, the relayed attestation model appears to be identical to the direct attestation model, because the client does not interact directly with the VSP. After accessing the SP's Web application (1), the SP requests a TNC check (2). If the TNC check request is accepted, a client sends integrity measurements to the SP (3). Instead of evaluating them, an SP acts as a transparent proxy and forwards all measurements to the VSP (4). Depending on the integrity policy stated by the SP, a VSP can request further measurements (5). The SP forwards this message to the client (6). After the TNC check is completed, the VSP can make an access recommendation to the SP (7). The SP evaluates this statement and enforces an access decision that is also sent to the client (8).

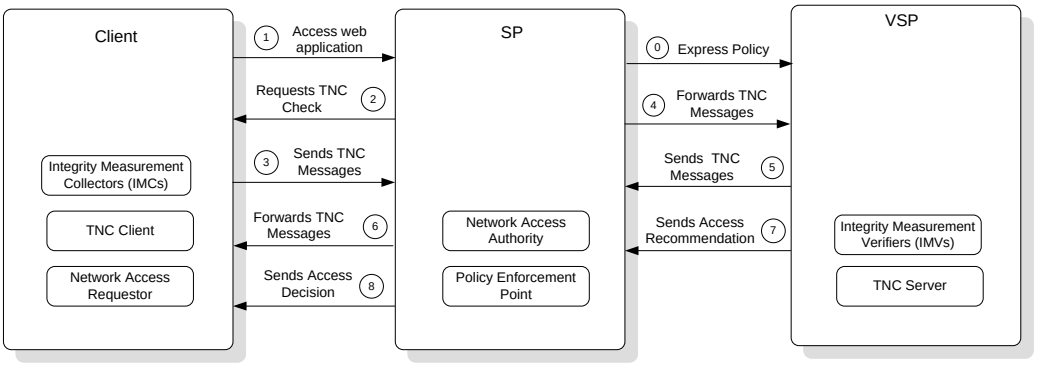


Figure 3.4: Message exchange in the relayed attestation model

As described in section 3.1, integrity measurements need to be sent encrypted. Because the SP relays all TNC messages between client and VSP, two communication channels exist. Each of these channels must be encrypted separately. This approach creates a performance overhead and a possible bottleneck at the SP because all messages have to be decrypted and re-encrypted before they can be forwarded.

As discussed above, an SP does not need to understand the TNC messages that are exchanged between client and VSP. Its only task is to relay them between client and VSP. However, a client cannot detect or prevent that an SP is eavesdropping on the exchanged messages. It is therefore still possible that the SP is able to determine details about the client's software configuration, which may raise privacy concerns. Furthermore, a malicious SP is able to derive attack strategies using details obtained through the integrity measurements. A model that addresses these privacy concerns and security risks is described in the next section.

3.3.3 Assertion-based Attestation

Similar to the model described above, the *assertion-based attestation* model consists of three independent parties. However, in this model the SP does not need to relay messages, as VSPs and clients are communicating directly. This approach is depicted in figure 3.5 and described in the following.

In the first step, a client accesses a service such as a web-based application provided by an SP (1). Before access to the requested resource is granted, the SP requests a TNC check (2). Similar to the relayed attestation model, an SP expresses its integrity requirements in form of a high level policy. This policy is sent to the user as part of the TNC check request.

When performing the TNC check with a VSP, the user forwards the integrity policy to the VSP (3). This approach allows a fine grained policy handling in which the integrity policy can be adjusted depending on the requested service and the client that accesses this service. After performing the TNC measurements (4), the VSP compares the obtained measurements with the integrity policy to derive an access recommendation. This recommendation is sent to the client (5), which forwards it to the SP (6). The SP uses the access recommendation from the VSP for making an access decision, which is sent to the client in the final step (7).

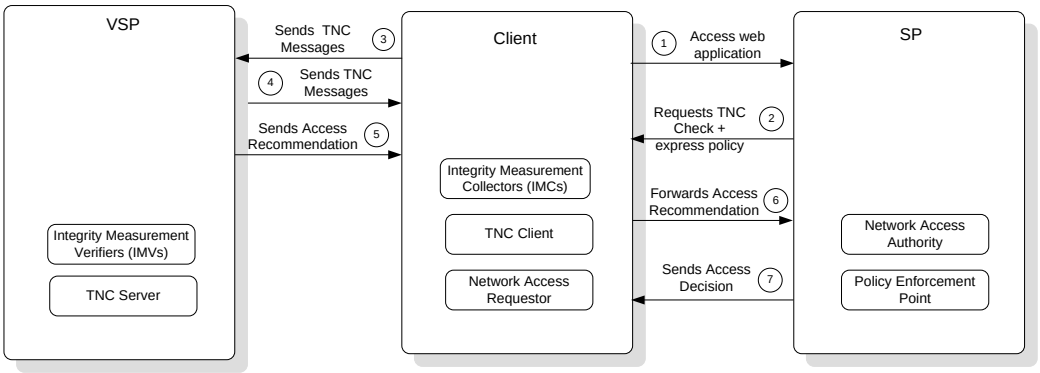


Figure 3.5: Message exchange in the assertion-based attestation model

The communication between SP and VSP is relayed through the client. To prevent a client from changing either the integrity policy or the TNC check result, these artefacts need to be integrity protected, that is, they must be secured with a digital signature. An alternative solution to this approach is to establish a back channel communication between VSP and SP in which integrity policies and TNC check results are exchanged. Such an approach, however, is associated with the following drawbacks:

Connectivity A model with back channel communication only works if the VSP can access the SP. It cannot be applied in situations, where the VSP and SP are not in the same network. An example for such a situation is an SP that is located in a private network. While a client that is part of the same network can access the SP, VSPs that are part of a public network, such as the Internet, cannot directly access the SP. Although private networks can be opened to a particular VSP by adjusting firewall rules, this approach becomes impractical when users can select from a large or dynamic group of VSPs. In contrast, the assertion-based model can handle such a scenario, as the client relays the communication between SP and VSP.

Complexity A back channel communication increases the complexity of the communication schema. Firstly, the SP has to accept connections from VSPs. This requires to open an additional interface, which needs monitoring and maintenance in order to prevent security breaches. Secondly, a back channel approach requires the SP to be stateful with regards to a TNC check. In order to be able to associate a TNC check result that is reported by a VSP for a particular user, the SP must retain information about the TNC check request and the associated user. Furthermore, clean up procedures must be created to remove state information for TNC checks for which no result was received within a certain time frame. Thirdly, as part of the state information, VSP and SP need to agree on an identifier or pseudonym for the user in order to be able to relate TNC results with a certain user. In addition to increasing the complexity, this approach may also raise privacy issues, as discussed next.

Privacy A back channel communication introduces two privacy issues. Firstly, the VSP needs to know the identity of the SP, for example in form of a URL, in order to be able to report the TNC check result to the SP. This approach enables a VSP to create a usage profile about the client by tracking which SPs a client uses. Secondly, as described above, it requires the VSP and SP to agree on a shared identifier for a client in order to associate a TNC check request with its result. Because the VSP is aware of the SPs identity, it allows a collaboration between VSP and SP in which detailed TNC integrity measurements can be linked to a client's identity.

In summary, the drawbacks related to a back channel communication prevent this approach from being applicable to web-based TNC checks. Although it requires digital signatures, the assertion-based attestation model is more versatile, less complex, and provides a better protection for the privacy of clients.

In general, a VSP does not require any client authentication. Offering the service without prior authentication prevents a user's identity from being linked to the software configuration of his or her PC. However, all publicly offered services are subject to misuse. For example, malicious users can mount Denial-of-Service (DoS)[1] attacks by performing fake integrity checks, which consume network bandwidth and CPU time. A possibility to mitigate the risk of such a resource depletion attack [SL04] is to require users to be authenticated. [Nik02] and [PLR07] conclude that authentication of IP messages can help to avoid DoS attacks at the network level. This approach can be used at a higher protocol layer to protect against DoS attacks at the application layer [SIYL08].

Two possibilities exist for the VSP to authenticate a user. In the first approach, the VSP performs the authentication itself. This requires users to have an account at the VSP and to login to the VSP before performing a TNC check. While such an approach is universally applicable, it adds another step in the process of performing a TNC check.

The second approach relies on the authentication that an SP performs. It is only applicable if a user performs an authentication with the SP before the SP requests a TNC check. In this case, the SP can issue a statement to a client that confirms a successful authentication. The client can use this statement to demonstrate its legitimacy to the VSP.

The method of authentication influences the underlying trust model of the assertion-based attestation model. In general, a user trusts a VSP for privacy purposes. An SP trusts a VSP to correctly perform a TNC check. If the VSP relies on an authentication statement from the SP, it is further necessary that the VSP trusts the SP. While such a trust relation is unlikely to occur in open environments, it can be established in closed environments were it is predetermined which VSPs a client uses. Authentication and trust models are further discussed in chapter 4.

3.4 Summary and Conclusion

In the previous sections, three attestation models have been proposed and compared. The direct attestation model is the simplest model and consists of only two parties. This model may raise privacy and security concerns, because detailed TNC integrity measurements are released to SPs. In addition, this model can be impractical to implement, as it requires the SP to perform the integrity check by itself.

[1] See http://www.cert.org/tech_tips/denial_of_service.html for more information on DoS attacks.

As a consequence, the relayed attestation model was developed. In this model, the VSP is realised as an independent party that performs the integrity check on behalf of the SP. The responsibility of an SP is reduced to formulating its integrity requirements in a policy. While this approach reduces the complexity for the SP, this model does not resolve the privacy and security issues of the direct attestation model. Furthermore, it introduces a bottleneck as all TNC messages have to be relayed through the SP.

The assertion-based attestation model addresses these issues by changing the communication flow. The measurement is performed between client and VSP without an interaction of the SP. Instead of details about a client's software configuration, an SP only receives a statement from the VSP that asserts whether the client conforms to the integrity policy stated by the SP. In the assertion-based attestation model, a client is expected to trust one or more VSPs. That is, the client trusts the VSP not to reveal integrity measurements that were gathered during a TNC check. Similarly, an SP trusts a VSP to perform a TNC integrity check according to the SP's integrity policy. This policy and the TNC check result are relayed through the client. Is is essential to protect the integrity of these assets, as a client could otherwise forge a TNC check result. A mechanism that achieves this is proposed in the next chapter.

Chapter 4

Asserting Integrity

The assertion-based architecture, as proposed in the previous chapter, provides the foundation for a web-based TNC check. This architecture modifies the flow of messages in TNC in order to adapt to the requirements of an open, web-based environment. In this architecture, a client performs the TNC check with a trusted VSP[1]. This VSP needs to inform an SP about the outcome of this check, that is, the VSP asserts the integrity state of a client to an SP. Such a mechanism is not defined within the TNC specification, where integrity check and access decision enforcement is performed by the same entity.

In this chapter, such a mechanism is developed. It enables a VSP to assert the integrity state of a client. In section 4.1 the requirements for this mechanism are summarised. Based on these requirements, section 4.2 describes how the mechanisms for expresing the integrity state of a client, as defined in the TNC specification, can be mapped to the assertion-based architecture. In section 4.3 a message format is proposed to encapsulate an integrity assertion in a standardised form. The message flow in the assertion-based architecture using the message format proposed in this chapter is presented in section 4.4. Finally, the results of this chapter are summarised and discussed in section 4.5.

4.1 Requirements

The purpose of this chapter is to find a mechanism that enables a VSP to propagate the integrity state of a client to an SP. The integrity assertion is relayed through the client and thus needs to be protected from being changed by the client. That is, the integrity of the assertion needs to be protected. A client, or a malicious program that is running on a client, would otherwise be able to forge the integrity check result.

Confidentiality and integrity protection The communication path between VSP and SP is interrupted by the client. As a consequence, transport layer protection cannot be used and the mechanism used for asserting the integrity check result has to support integrity protection at message level.

[1] Cf. figure 3.5.

The messages exchanged during the TNC check and the TNC check result contain sensitive information. For example, the messages exchanged during the TNC check can contain information about which security patches have been applied or which version of a particular software is installed. If an attacker has access to this information, he or she can potentially derive and exploit vulnerabilities in software that is executed on the client machine. It it thus necessary to protect the confidentiality of the exchanged message.

Message recipients Both parties, the client and the SP, need to be informed about the TNC check outcome. An SP needs to know the outcome of the integrity check and the underlying integrity policy in order to determine whether to grant a client access to its services. The client needs to know the outcome in order to trigger a remediation procedure if it is not compliant to the SP's integrity policy.

Open standards The assertion-based architecture does not require an SP to know details about the TNC integrity check. Instead, an SP is only required to trigger a TNC check and to assess the result. By hiding the details of the TNC check from the SP an integration into existing authentication and authorisation systems is simplified. To achieve this, the open design principle [SS75] is applied and open standards and existing mechanisms are used preferably.

The following section describes how the TNC integrity check result can be mapped to the assertion-based architecture. Section 4.3 discusses how this mechanism can be encapsulated in a message format.

4.2 TNC Recommendations in the Assertion-Based Architecture

The result of a TNC check is an access recommendation based on the TNC measurements and policies stating the required integrity state. As stated in section 4.1, this recommendation needs to be sent to two parties, that is, the client and the SP. In the original TNC architecture, a single entity performs the TNC measurement and creates the final access decision. In the process of adapting the TNC architecture to a web-based environment, this entity is divided into two entities, as discussed in chapter 3. The VSP is dedicated to perform all TNC measurements and has to report the outcome of the TNC check to the SP. It is within the responsibility of the SP to make and enforce the final access decision. This section analyses how the existing TNC mechanism of expressing an access recommendation can be mapped to this scenario.

As described in section 2.2, IMV components collect TNC measurements and compare them to an integrity policy. Based on this comparison, they have to produce two statements for the TNC Server (TNCS): An *IMV Action Recommendation* [Tru07d, section 3.4.2.5] and the

Table 4.1: Mapping of TNCS Action Recommendations to Policy Conformance Statements in a web-based TNC environment

TNCS Action Recommendation	Policy Conformance Statement
Allow	Compliant
Isolate	Non-Compliant
None	Indeterminate

IMV Evaluation Result [Tru07d, section 3.4.2.6]. The latter states whether a client is compliant with the integrity policy and, if it is not compliant, whether the deviation from the policy is minor or major. Furthermore, an IMV can also state that it was not able to determine the client's compliance with a policy. Based on the evaluation result, an IMV recommends an action to be taken by the TNCS by proposing one of four access levels in an Action Recommendation [Tru07d, section 3.4.2.5]. If a client complies with the policy, the IMV recommend allowing access. Otherwise it can suggest to deny access or to isolate the client in order to start a remediation process. In all other cases, for example if an error occurred, an IMV can state that it cannot provide a recommendation. Optionally, an IMV can also provide a *Reason String* in which it indicates why a certain recommendation was made.

A TNCS collects all IMV statements and combines them to form a *TNCS Action Recommendation*[2]. Similar to the action recommendation provided by IMVs, a TNCS action recommendation can suggest one of three access levels: Allow, isolate, or none. The TNCS sends this Action Recommendation to both the NAA and the TNC Client (TNCC). The NAA will create the final access decision based on this recommendation and will then inform the enforcement point about its decision.

The recommendation sent to the client-side TNCC is encapsulated in a TNCCS Batch[3]. This message type is also used for encapsulating integrity measurements exchanged during the TNC message exchange. The format of the recommendation does not need to be altered for the assertion-based attestation architecture. After completing a TNC check a TNCS can send the access recommendation to the client using a standard TNCCS message.

In the original TNC architecture, the TNCS and NAA are part of the same entity, that is, the Policy Decision Point. In the assertion-based architecture however, as described in section 3.3.3, TNCS and NAA are realised in two different entities. Conceptually, the information that needs to be sent from the TNCS to the NAA remains the same. A TNCS combines all IMV Action Recommendations into one TNCS Action Recommendation which is sent to a NAA. However, three differences need to be addressed. Firstly, instead of a local function call that returns a value, the action recommendation now needs to be sent over a public network. Secondly, the TNCS Action Recommendation values (allow, isolate, none) suggest that the VSP provides an authorisation statement. However, from an SP's viewpoint,

[2]Cf. figure 2.2.
[3]See section 2.2.

the VSP issues a statement about the integrity state of a client and its conformance to an integrity policy. To reflect this, the TNCS Action Recommendation values are mapped to values that reflect the SP's point of view, as table 4.1 shows. Finally, TNCS and NAA cannot communicate directly. Instead, the VSP issues an assertion about the client to the SP using the client to relay the message. The encapsulation mechanism for this message needs to address this and must prevent clients from changing the TNC check result.

In the next section, an encapsulation mechanism for the TNC check result that is based on open standards and takes these requirements into account is proposed.

4.3 Encapsulation of the TNC Result

An SP can use the TNC check result to adjust its authorisation decisions. This enables the TNC result to be used as an additional attribute in an authorisation process. This approach is particularly straight forward in Attribute-Based Access Control (ABAC) systems [PDMP05, YT05, KSP07, SG07], in which access control decisions are purely based on attributes of an access requesting party. Examples of these attributes are associated roles, rights to access a resource, or age. The TNC check result can also be considered as an attribute in an ABAC-based system. However, the concept of attributes also exists in other approaches. [SSMP06] surveys several authentication and authorisation systems and identifies two standardised techniques that are used for exchanging attributes: X.509 Attribute Certificates and the Security Assertion Markup Language (SAML). The following sections discuss if and how these techniques can encapsulate the TNC check result in the assertion-based architecture.

4.3.1 X.509 Attribute Certificates

An X.509 *public key certificate (PKC)*[4] [Int05] binds an identity to a public key. A PKC can be used to provide evidence of a person's identity. To prevent forgery, a PKC is digitally signed by a *Certification Authority (CA)*. As PKCs tend to have a long lifetime, they are unsuitable to carry attributes that are likely to change (such as the TNC check result). If one attribute needs to be revoked, it is necessary to revoke the whole PKC [FH02].

To overcome this limitation, *attribute certificates (AC)* have been introduced as part of the X.509 specifications [Int05]. An AC can contain attributes of the AC holder, such as associated roles, group memberships, or authorisation information [FH02]. Similar to a PKC, an AC is signed by a trusted party to prevent forgery. This party is called an *Attribute Authority*. In contrast to a PKC, an AC does not contain a public key. Instead, it contains a reference to a PKC.

[4] Also called an identity certificate [BLMT04].

An example, based on [FH02], clarifies the relation of PKCs and ACs. A PKC is similar to a passport: it identifies its holder, has a long validity period, and is hard to obtain. An AC can be compared with an entry visa. It is typically bound to a passport, issued by a different authority that issued the PKC, and has a shorter validity period.

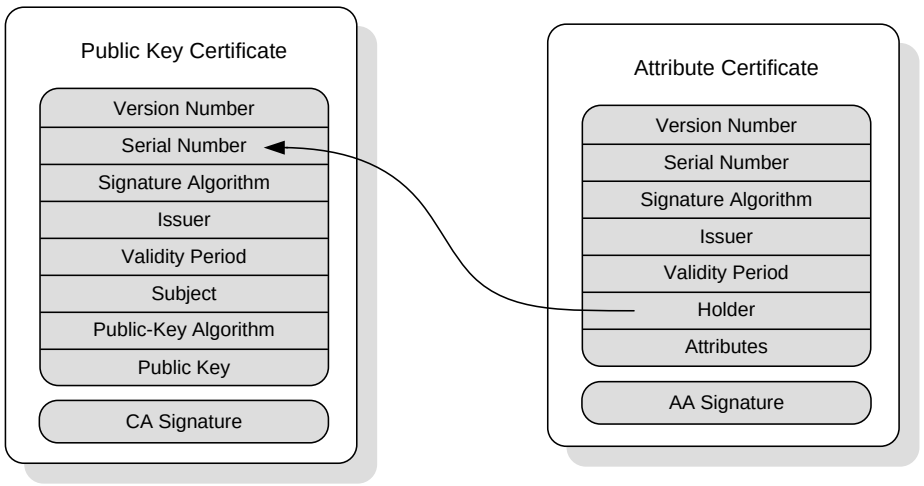


Figure 4.1: Relation between Public Key and Attribute Certificate

Figure 4.1 shows the structure of and relation between PKCs and ACs. The PKC contains (among other information) an identifier of the subject, the subject's public key, and a unique serial number which identifies the certificate. The *holder* of an AC uses the unique serial number of a PKC to refer to a particular PKC and thereby to a particular subject and its key. The *attribute* field can be used to include attributes that are associated with the holder of the AC. In addition to simple name/value pairs, ACs also support attributes that are expressed using an XML structures [Int05, section 14.5]. This flexibility in expressing attributes is sufficient for encoding the TNC check result and thus to include it in an AC.

However, the association of an AC with a PKC and therefore a public and private key would complicate a TNC check based on ACs. Private keys must be handled with particular care and stored in a protected storage, as a stolen private key can be used to impersonate its legitimate owner. Techniques such as TPMs[5] or smart cards[6] can be used in order to store key material in a protected way.

However, even if the key handling is secured, using ACs (and thereby PKCs) would add further complexity to a web-based TNC. An identity certificate, such as a PKC, can be used to prove the identity of its owner for authentication purposes. The use of ACs requires the use of PKCs for user authentication. PKCs are managed within a *Public Key Infrastructure (PKI)*. One property of such a PKI is that all participants are able to validate identities using

[5] Cf. section 2.4.

[6] Cf. [RE04] for an extensive overview of smart cards and their capability to provide a secure storage for key material.

PKCs because the same CAs are trusted by all participants (either directly or indirectly). If ACs and PKCs are used in the assertion-based architecture, SPs and VSPs must thus be members of the same PKI in which PKCs are issued for end users. However, this contradicts the purpose of the assertion-based architecture, that is, supporting an open environment instead of a closed and managed environment, such as a PKI. Furthermore, the use of a PKI threatens the privacy of users because a unique identifier (the public key contained in a PKC) is used to refer to users by all participating parties. Work has been done to improve the privacy situation in systems that rely on ACs [BLMT04, AVKK+04]. However, these approaches further complicate a solution based on ACs and PKCs [BLMT04] and introduce non-standardised certificate formats [AVKK+04].

Moreover, the use of a unique identifier for a user is not required for a web-based TNC check. It is not necessary that SP and VSP agree on a shared identifier for a user. In contrast, a client is able to present its integrity check result to the SP without revealing identity information to the VSP.

Introducing PKCs to a web-based TNC check increases the complexity unnecessarily as users need to be issued with PKCs. To avoid adding this complexity, a more lightweight approach is necessary that does not rely on user certificates. Such an approach is discussed in the following section.

4.3.2 Security Assertion Mark-up Language

The *Security Assertion Mark-up Language (SAML)* [SC05] defines an XML-based framework for exchanging security assertions. A security assertion is used to transfer information about a subject (such as identity, attributes, or entitlements) from one entity to another. SAML 2.0 was ratified as an OASIS[7] standard in 2005 and is considered the *de facto* standard for exchanging security assertions in heterogeneous environments [PA07]. SAML intents to solve problems in the following use cases [SAM08]:

Web-based Single-Sign On (SSO) The aim of an SSO system is to allow a user to use several independent services or applications while only performing one authentication. In web-based systems, authentication often relies on passwords [Fur07]. An SSO mechanism reduces the need for a user to remember a username/password combination for every single application. In a typical SAML SSO scenario, a user authenticates to an entity called the *Asserting Party (AP)*. This entity asserts information about the user. For example, it can assert that the user has been authenticated using a password mechanism, his name is Joe Smith, and his email address is js@example.com. This information is encapsulated in a SAML assertion which is digitally signed by the AP.

[7] The *Organization for the Advancement of Structured Information Standards (OASIS)* is an consortium that develops and maintains specifications and standards in the field of Web services. More information about OASIS can be found at http://www.oasis-open.org.

If a user accesses a service, he or she can provide this assertion to the service provider instead of performing a separate authentication. In SAML, a party that offers a service and accepts SAML assertion is called a *Relying Party (RP)*. The SAML SSO mechanism requires that the RP trusts the information that an issuer has placed into the SAML assertion.

Federated Identity In a federated identity system, several online services collaborate and use a common user base. In these scenarios, users often have an individual local user identity associated with one partner. Identity federation provides a mechanism for these partners to share information about a user across organisational boundaries. A common example use case for federated identity is built around a travel booking scenario [RR04, Sul05, SAM08]. In this scenario, a user needs to make travel arrangements, such as booking a flight, accommodation, and a rental car, using one service provider. In this example, the airline service used to book the flight acts as this service provider. Through agreements with rental car services and accommodation providers, the airline can book further services for the user by using assertions. The advantage of this approach is that a user only has to authenticate once with the airline service. The airline service will take care of establishing a shared name identifier for the user among the other service providers. This name identifier can be used to refer to an identity, for example for payment purposes. A user has a federated identity if such an agreement about how to refer to a user was established. SAML provides methods to manage such a federated identity scenario.

Security Token for Web Services SAML assertions can also be used as a security token in other technologies, such as SOAP-based Web services[8]. Instead of relying on passwords or X.509 certificates, a SAML assertion can be used to authenticate a SOAP request [Mon06]. One advantage of this approach is that it allows attributes about the assertion subject to be included within the security token.

The core concept, which is used in all three use cases, is the notion of SAML assertions. They are used to transfer security information from one entity to another. As such, they can be used to transfer the TNC check result from the VSP to the SP. SAML assertions are described in more detail in the next section. Integrity protection of SAML assertions is covered in section 4.3.2.2. Based on these fundamentals, section 4.3.3 proposes a message format to use SAML for encapsulating the TNC check result.

4.3.2.1 Assertions

Assertions are the central concept in SAML. An assertion contains statements about a subject (typically a user). Three different types of statements are defined in SAML. They are described below [SAM08, SC05]:

[8] Cf. section 5.3.2.

Authentication statements are created by the Asserting Party to state that a subject was successfully authenticated. Authentication statements include information about *who* was authenticated *when* using which *authentication method*.

Attribute statements contain qualifying attributes about the subject, for example about his or her age or a credit limit. This type of assertion will be used to encode the TNC result that reflects the conformance of a client to an integrity policy.

Authorization Decision statements contain information that defines what actions a subject can perform on a resource. For example, such a statement can contain the information that *Joe Smith* has read access to a certain document identified by its URI.

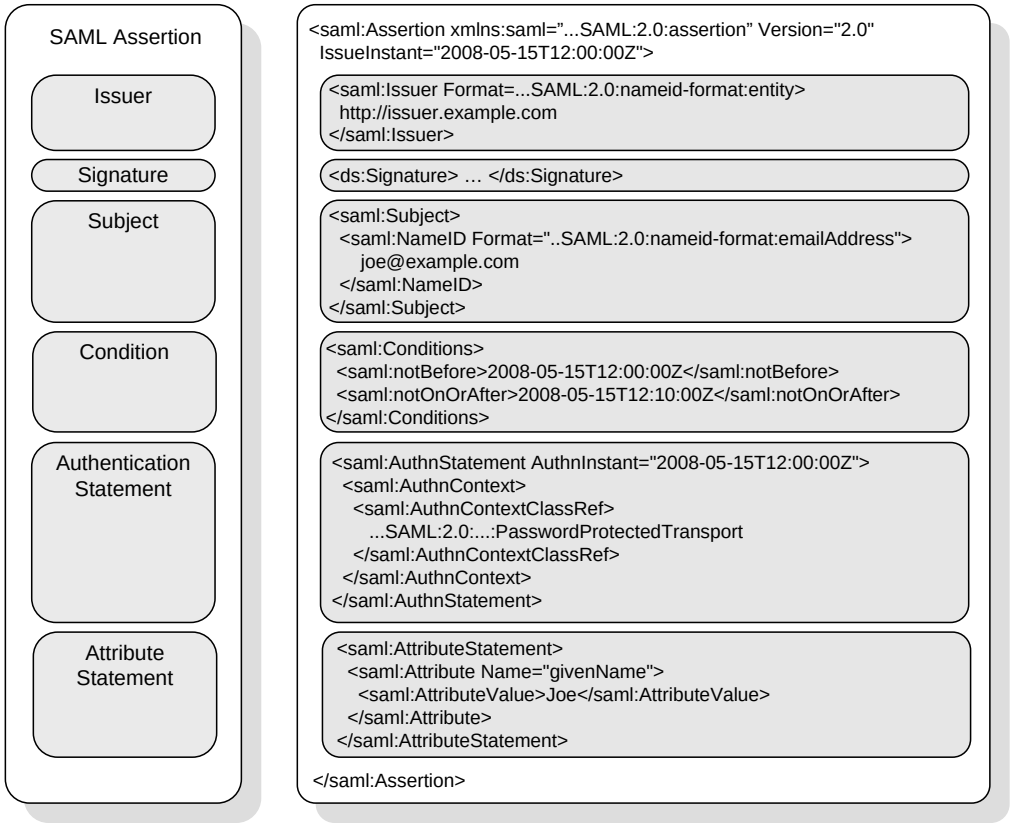


Figure 4.2: Example of a SAML assertion carrying an authentication and attribute statement

Figure 4.2 outlines a SAML assertion and shows its XML representation. A SAML assertion can only be accepted by a relying party if it trusts the asserting party that has issued the assertion. It is therefore necessary to include information about the asserting party's identity into the assertion. Two elements contain information for identifying the asserting party. Using an identifier such as a URI, the element *Issuer* indicates the identity of the asserting party. Furthermore, a digital signature is provided in the element *ds:signature*. In addition to preserving the integrity of the assertion, this elements typically points to a X.509 certificate that contains a key to validate the signature. Using this certificate, the relying party

can identify the asserting party. This approach is described in more detail in section 4.3.2.2. Furthermore, an assertion can also contain an identifier for the subject of the assertion (using the element *Subject*). In the example the subject is identified using an email address. Typically, an assertion can only be used for a limited amount of time. The validity period (10 minutes in this example) can be expressed using the *notBefore* and *notOnOrAfter* statements in the *Conditions* block. As described above, an assertion contains statements about the subject. In this example, the assertion contains an authentication and an attribute statement. The authentication statement evinces that the subject was authenticated at 12pm on the 15th of May 2008 using a transport security protected and password-based mechanism. The attribute statement below contains one attribute, stating the given name of the subject.

SAML assertions are usually passed from one party to another. It is therefore essential that the content of the assertion is protected from being altered. In the assertion-based architecture, an assertion containing the TNC result will be issued from the VSP and is relayed through the client to the SP. Because the assertion is relayed, its integrity must be protected at the message level. The following section describes how SAML assertions can be integrity protected using XML Signature.

4.3.2.2 Integrity Protection of SAML Assertions

In the assertion-based architecture, VSP and SP do not communicate directly. Instead, all communication is passed through the client. Transport security is terminated at the client, requiring protection at the message level. During the communication, the VSP sends the TNC check result to the client assuming that the client will forward it to the SP. To prevent clients from changing the value of the check result, the message needs to be integrity protected, as stated in section 4.1.

SAML assertions can be signed using an XML Signature as defined in [SC05]. *XML Signature (XML-DSig)* [BBF+02] defines an XML-based syntax that allows to digitally sign XML documents. An assertion signed by the asserting party provides assertion integrity and authentication of the asserting party. The structure of an XML Signature is described in more detail in the following.

XML Signature XML Signatures are specified as a W3C[9] recommendation. They allow digital signature technology to be applied to XML documents. An XML Signature is itself expressed in XML. Listing 4.1 shows the structure[10] of such an XML Signature statement.

[9] The World Wide Web Consortium (W3C) is an international consortium devoted to developing Web standards. See http://www.w3.org/Consortium/ for more information.

[10] The structure is presented using the XML Shorthand Schema notation [RR04, page 114] as used in [BBF+02]. In this notation, the character "?" denotes an optional element (zero or one occurrence), "+" denotes an element with one or more occurrences, and "*" denotes an element with arbitrary occurrences (zero or more).

```
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    (<Reference URI? >
      (<Transforms>) ?
      <DigestMethod>
      <DigestValue>
    </Reference>)+
  </SignedInfo>
  <SignatureValue>
  (<KeyInfo>) ?
  (<Object ID?>)
</Signature>
```

Listing 4.1: Structure of an XML Signature statement

The *Signature* element is the root element of an XML Signature, that is, all other elements are children of *Signature*. A signature must have exactly one *SignedInfo* element. Its sub-elements contain information about how the signature was computed.

Creating a digital signature requires two steps. In the first step, a cryptographic one-way function (for example SHA-1) is used to create a hash value that reflects the data that is to be signed. In the second step, this hash value is combined with a cryptographic key (for example using the RSA algorithm) to create the signature of the message. If one bit in this message changes, the resulting hash value will change and so will the signature. This poses a problem for XML documents, as an XML document can be represented in several (physical) ways without changing its meaning. An example is the usage of different line ending character sequences in different operating systems. A file created on an Windows operating system will typically have a different ASCII character sequence to indicate a line ending (\r\n) from a file created on a Unix-based operating system (\n). To overcome this issue, *canonicalisation* is used. Canonicalisation[11] *normalises* the XML document so that, albeit physical differences in two logically equivalent documents, their physical bit-by-bit representation will match, thus producing the same signature value. The *Canonicalization-Method* states which transformations have been applied to the *SignedInfo* element before the signature was computed.

The algorithm that is used for generating the digital signature is defined by *SignatureMethod*. Commonly used examples are DSA with SHA1 and RSA with SHA1 [BBF+02, section 6.1]. *Reference* contains a pointer (in form of a URI) to the data that is signed. The *Transforms* element contains information about which transformations, for example a canonicalisation method, have been performed with the data to be signed prior to calculating the one-way

[11] Sometimes abbreviated as *C14N* – 14 letters enclosed by *"C"* and *"N"* [RR04, page 120].

hash value. *DigestMethod* states which hashing algorithm was used to calculate this hash value that is held in the *DigestValue* element.

The *SignedValue* element contains the digital signature of the *SignedInfo* block. By signing this block, the signature does not only protect the referenced data, but also the methods used to generate the signature. The optional *KeyInfo* element contains information that enables the recipient of a signature to obtain the key that is necessary to verify the signature. This element can either contain the key itself (e.g. in form of an X.509 certificate) or information that enables the recipient to look up the key (such as a key name or thumbprint).

XML Signature supports three different modes for relating a signature to the signed data: *enveloping*, *enveloped*, and *detached*. They differ with regards to the location of the data that needs to be signed. Enveloping signatures sign data contained in the *Object* element, enveloped signatures sign the content of their parent XML elements, while detached signatures sign information that is located outside the signature containing document. The SAML specification defines that only enveloped signatures can be used for SAML assertions [SC05, section 5.4.1].

Generating and Verifying of XML Signatures In order to generate an XML Signature, two steps are necessary. Firstly, generating the *SignedInfo* block and secondly generating the dependant *SignatureValue* block [BBF+02]. For creating the SignedInfo block, a hash value for each canonicalised data block that needs to be signed is computed. After the *SignedInfo* block has been created, a hash of this XML structure is calculated and the signature can be created using appropriate key material.

The verification process must be performed in reverse order to validate the signature. It is divided into two steps: Reference validation and signature validation [RR04, page 119]. Validating the reference ensures that the elements being pointed at by the references have not been changed. The required canonicalisation method is applied to the *SignedInfo* element. For each *Reference*, the digest of the canonicalised XML structure is created and compared with the value in the *Reference* block. If all values match the computed digests, the next step, that is signature validation, can be performed to ensure that the *SignedInfo* block was not tampered with and to validate the identity of the signature creating party. Using the *KeyInfo* element, the verifier can obtain the key for verification. For example, this can be the public key contained within a X.509 certificate. The required canonicalisation method is applied to the *SignedInfo* element and the element's hash is calculated. This hash is now compared to the decrypted *SignatureValue* obtained by using the verification key. If these values match, the signature was validated successfully and the integrity and authenticity of the message has been verified[12].

[12] This verification can be trusted to the extent that the verifier trusts the certificate issuing CA (if a certificate was used) and that the key used to create the signature was kept securely.

4.3.3 Using SAML for Expressing the Integrity Check Result

The previous sections have provided the building blocks for using SAML to express a TNC check result. Firstly, sections 4.1 and 4.2 summarises which information must be asserted by a VSP to an SP. Furthermore, section 4.2 also describes a mapping of TNCS Action Recommendation produced by a TNC stack to policy conformance statements expected by an SP. In section 4.3.2.1 SAML assertions have been described and the idea of using SAML attribute statements to encapsulate the TNC check result has been proposed. Finally, section 4.3.2.2 summarises how to prevent forgery and manipulation of SAML assertion using XML Signature. This section assembles these building blocks and proposes a SAML-based message format for encapsulating the TNC integrity check. This message format is illustrated using a simplified example that is given in listing 4.2. This examples, which contains all the elements previously discussed, is examined in the following[13].

The SAML assertion starts in line 1 stating a unique ID and the time of issuance. The issuer in line 2 is identified using its URL. The assertion is signed using an XML Signature (lines 3–18). The *Reference* element (line 7) points to the root *Assertion* element, thus protecting the integrity and authenticity of the whole assertion. An X.509 certificate is supplied in the *KeyInfo* block for checking the identity of the issuer of the assertion (lines 13–17).

A SAML assertion can include information that identifies its subject. Based on the subject's identity, a SAML assertion can also be bound to this subject. This process is called subject confirmation and is based on proving knowledge of a private asymmetric key. However, such an approach is not applicable to the web-based TNC check, as VSP and SP do not share an identifier for a client. SAML defines the *bearer* subject confirmation method for such a situation, so that the subject is allowed to be anonymous. This is expressed in lines 19–21.

The SAML assertion includes an attribute statement in lines 22–38. This attribute statement contains an attribute called "TNCResult". The value element for this attribute contains the proposed custom XML structure for carrying the TNC check result (lines 25–35). This structure maps all required information to an XML syntax. As described in the requirements in section 4.1, the TNCS produces a TNCS Action Recommendation stating whether the client conforms to a given policy. This is reflected in line 26 using the element *TNCIntegrityCheckResult*. The TNCS collects optional reason strings produced by IMVs to explain their action recommendations. Lines 27–29 provide a structure that allows to express this.

The VSP uses an integrity policy to derive the integrity check result. As covered in detail in chapter 6, the SP states which integrity policy a VSP has to use for deriving the integrity state of a client. It is essential that an SP can be certain about which policy was used to derive a certain TNC check result. In general, there are two possibilities to achieve this. Firstly, the SP could digitally sign the integrity policy. This would prevent a client from changing

[13] The XML Schema of the proposed format can be found in section D.1.

the policy to its favour and enable an SP to verify the policy's integrity and authenticity. By trusting the VSP to perform this validation, the SP can correlate an integrity policy with an integrity check result. However, this approach requires that the VSP has access to the SP's public key certificate, which contains the key to validate the signature of the policy. This approach is not feasible in an open, web-based scenario, in which the relation between an SP and supported VSPs is not predetermined. An alternative to using an additional public key certificate is to send the policy, or an representation thereof, back to the SP. In this approach, which is used in the following, both the policy and the integrity check result are signed by the VSP. The SP can therefore verify that the policy which arrived at the VSP and which is used to derive the integrity check result is the same policy as the one which was sent to the VSP.

As outlined later[14], an SP has two possibilities of expressing a policy. It can either state it explicitly using its XML representation or refer to it using an identifier (such as a name, an ID, or a URI). This is reflected in the *wTNCIntegrityPolicy* statement in lines 30–34 which include these policy identifying attributes. If the SP states its integrity policy explicitly, the VSP will include a hashed representation of this policy to reduce the size of the SAML assertion[15]. As already described in section 4.3.2.2, an XML document can have several physical appearances that all have the same logical equivalent. However, the hash value computed for each of these physical appearances will differ. The same solution that is applied to this problem in XML Signature can also be applied here. The integrity policy must be canonicalised before its hash value is computed. Different canonicalisation and hash algorithms exist. To avoid limiting this process to a particular set of algorithms, the semantics of XML Signature are adopted to indicate which algorithms have been used for generating the hashed policy representation (line 31–33). This ensures that the SP can validate the hash and thus validate that the integrity policy has not been tampered with.

Finally, the SAML assertion includes a *Condition* statement that limits the validity period of the assertion. This mechanism can enforce the freshness of a SAML assertion and prevents that a client re-sends an expired version of an assertion.

The format proposed above is used to carry a TNC check result within a SAML assertion. This SAML assertion is part of the message exchange between VSP, client, and SP. This message exchange is described in the following section.

[14] Cf. chapter 6.

[15] The hashed representation of the sample policy given in section 6.3.3.2 has a size of 371 bytes. The size of its hashed representation is only 217 bytes reducing the size to approximately 58% of its original size.

```
<saml:Assertion Version="2.0" ID="_cd8854..."
    IssueInstant="2007-12-28T05:00:00Z"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:wTNC="http://www.canterbury.ac.nz/wTNCResult#">
 <saml:Issuer>vsp.example.com</Issuer>
 <ds:Signature>
  <ds:SignedInfo>
   <ds:CanonicalizationMethod Algorithm="http://www.w3.org/.../xml-exc-c14n#"/>
   <ds:SignatureMethod Algorithm="http://www.w3.org/.../xmldsig#rsa-sha1"/>
   <ds:Reference URI="#_cd8854...">
    <ds:DigestMethod Algorithm="http://www.w3.org/.../xmldsig#sha1"/>
    <ds:DigestValue>TCDVS...rxIPE=</ds:DigestValue>
   </ds:Reference>
  </ds:SignedInfo>
   <ds:SignatureValue>xjGCjwcRCK...vdVNCcY5=</ds:SignatureValue>
   <ds:KeyInfo>
    <ds:X509Data>
     <ds:X509Certificate>wPogg4rN...</ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
  </ds:Signature>
  <Subject>
   <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"/>
  </Subject>
  <saml:AttributeStatement>
   <saml:Attribute Name="TNCResult" NameFormat="...">
    <saml:AttributeValue>
     <wTNC:TNCResult>
      <wTNC:TNCIntegrityCheckResult>compliant</wTNC:TNCIntegrityCheckResult>
      <wTNC:TNCCSReasonStrings>
       <wTNC:ReasonString xml:lang="en">Reason for recommendation</wTNC:Reason>
      </wTNC:TNCCSReasonStrings>
      <wTNC:wTNCIntegrityPolicy id="..." name="..." URI="...">
       <ds:Transforms><ds:Transform Algorithm="...xml-c14n11"/></ds:Transforms>
       <ds:DigestMethod Algorithm="...xmldsig#sha1"/>
       <ds:DigestValue>TCDVS....GrxIPE=</ds:DigestValue>
       </wTNC:wTNCPolicy>
     </wTNC:TNCResult>
    </saml:AttributeValue>
   </saml:Attribute>
  </saml:AttributeStatement>
  <saml:Conditions>
  <saml:notBefore>2007-12-28T05:00:00Z</notBefore>
  <saml:notOnOrAfter>2007-12-28T05:10:00Z</notOnOrAfter>
 </saml:Conditions>
</saml:Assertion>
```

Listing 4.2: Example of a simplified SAML 2.0 Assertion with an Attribute Statement used for encapsulating a TNC check result

4.4 Communication Model

In chapter 3, the assertion-based attestation model was developed for performing a TNC-based integrity check. Section 4.3.2 identifies SAML as a mechanism capable of encapsulating the TNC check result and proposes an XML-based structure that reflects the requirements stated in section 4.1 and 4.2. This section shows how this SAML assertion can be used in the assertion-based architecture. As mentioned in section 4.1, the TNC integrity check shall be performed as part of an existing authentication mechanism. The proposed approach is based on existing open standards that simplifies the integration into existing systems. This approach is depicted in figure 4.3 and is now described.

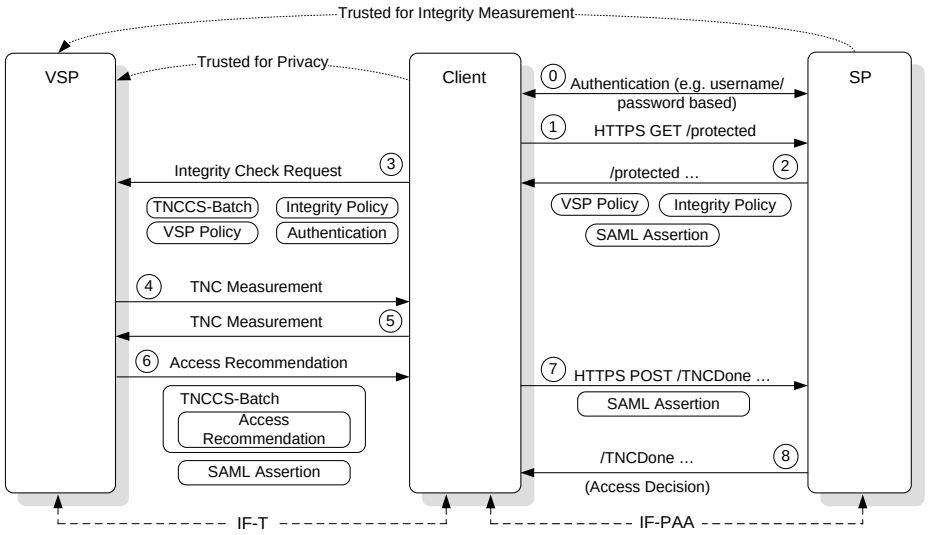


Figure 4.3: Message Flow in the Assertion-based Attestation Model

Performing a web-based TNC check increases the confidence of an SP about a client's security state and reduces the risk of malicious software interfering with the Web application. Before an SP requests a TNC check, a client must therefore attempt to access a functionality of a web-based application that requires this level of confidence. It can be expected that such a functionality is not offered without prior authentication of the user. This is reflected in step 0 in the communication model, in which the client is authenticated by the SP using any n-factor authentication method, for example, a username and password based mechanism. Such an authentication will typically be performed using an SSL/TLS secured HTTP (HTTPS [Net00]) transport mechanism.

After being successfully authenticated, the user attempts to access a protected area of the Web application which requires a TNC check (step 1). Instead of delivering the requested Web page, the SP sends a response in step 2 that triggers the TNC integrity check. The

delivered Web page includes the integrity policy[16], an optional authentication statement as described in section 3.3.3, and a VSP policy.

Using the VSP policy, an SP can state which VSPs are trusted for performing the TNC integrity check on the SPs behalf. The VSP policy format[17] allows two approaches for specifying this information: explicit and implicit. In the explicit case, a VSP assembles a list of trustworthy VSPs and their correspondent URL at which the TNC service is offered. Optionally, an SP can also supply an X.509 certificate for this VSP's public key (which could be signed either by a trusted CA or by the SP itself). The use of an X.509 certificate enables the client to check the VSP's public key when establishing the HTTPS connection, and protects it from being redirected to malicious verifiers. However, this approach requires that the client is in possession of a key to validate the certificate. It further requires that the SP updates the supplied certificate every time the VSP changes its SSL/TLS certificate.

In the implicit use case, the trust in a certain VSP is based on its capabilities. The trust in a certain VSP is established through the CA that issued the certificate whose associated private key is used to sign the SAML assertion. As described previously, integrity protection of the SAML assertion is provided by a signature. This signature is generated by using a key that is bound to a certificate. This certificate is issued by a CA, which is trusted by the SP to only issue certificates to VSPs that perform the TNC check to a certain standard. The agreement about these standards is not part of the actual TNC check. Using the VSP policy, an SP can state that it trusts all VSPs that are certified by a certain CA. This CA issues a certificate to a VSP as a proof of the certification process. When receiving a TNC check result encapsulated in a SAML assertion, an SP can check whether the issuer of the certificate which is used to obtain the public key to validate the signature is trusted. It is worth pointing out that this scenario requires a PKI scheme. However, unlike the approach based on Attribute Certificates described previously, this approach does not require the client to be part of this PKI scheme.

As with other services offered in the Internet, providing the TNC service without prior authentication presents a security risk [FZML02]. A security token, for example a SAML assertion created by the SP, can be used to provide confirmation to the VSP that the user has been authenticated. This assertion contains an authentication statement asserting method and time of authentication and provides a Single-Sign-On environment for the user. Alternatively, the user can perform an authentication with the VSP. This requires the user to have a user account with the VSP.

In step 3, a TNC client software, implemented as a browser plug-in, picks up the integrity check request. The client chooses a VSP that fulfils the requirements stated by the SP and is trusted by the user to protect its privacy. The TNC client software contacts the chosen

[16]The integrity policy is expressed in an XML-based format and is stating the integrity requirements for the client. See section 6.3.3.2 in which details about this format are described.

[17]Cf. section 6.3.3.3 for a detailed description of the VSP policy format.

VSP via a TLS/SSL protected channel to start the integrity check. The initial integrity check request sent by the client to the VSP contains the SAML assertion and the policy files. After the SAML assertion has been validated, the TNC measurement is performed in steps 4 and 5 according to the TNC specifications[18].

After the VSP has determined whether the client's security state is compliant with the policy received from the SP, a TNCS Access Recommendation is sent to the client (step 6). This access recommendation is encapsulated in a TNCCS Batch and can be processed by the client's TNCC software. The VSP creates and signs a SAML attribute assertion containing the result of the integrity check, as described in section 4.3.3. If the client is not compliant with the SP's policy, it can now perform a remediation process and restart the integrity check. Otherwise, it extracts the SAML assertion from the TNCCS-Batch and forwards it to the SP (step 7).

The SP verifies the signature of the SAML document (using the VSP's public key) and extracts the access recommendation. Based on this recommendation, it can then enforce the access decision, that is, allow or deny access to the operation requested by the client in step 1. The TNC check is now finished and further application data can be exchanged (step 8).

4.5 Summary and Conclusion

This chapter proposes a message format for the TNC check result based on a SAML assertion and a communication model in which this assertion is used. For the message format, two mechanisms that are commonly used in authentication and authorisation systems have been analysed. An X.509 Attribute Certificate based approach, described in section 4.3.1, introduces management and handling overhead, as it requires X.509 Public Key Certificates for all users. While a SAML assertion can also make use of X.509 certificates in XML signatures, clients (in the assertion-based architecture) are unaware of these certificates. Section 4.3.3 proposes an XML-based format for encapsulating the TNC check result using SAML attribute assertions.

These assertions have been integrated into the assertion-based architecture to form a communication model. This model provides a flexible approach for enabling TNC in web-based environment. TNC results are encapsulated in SAML assertions, which are passed to the authentication and authorisation system of the SP, which can then derive an access decisions. The SP can express its integrity and VSP requirements in policies.

Optionally, the proposed model supports an SSO mechanism for authenticating a user at the VSP. If the client uses a SAML assertion issued by the SP to perform an authentication,

[18] Cf. [Tru07c, Tru07d] and section 2.2.

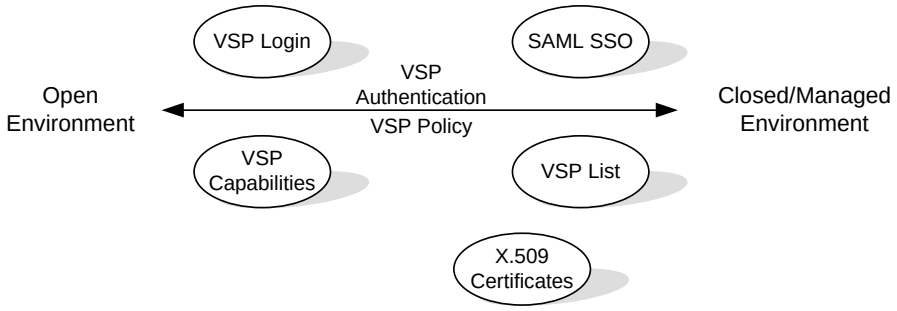


Figure 4.4: Allocation and relation of VSP policy and VSP authentication methods in different environments

the VSP is aware of the SP's identity as this SAML assertion needs to be signed by the SP. This signature needs to contain means to identify the SP, such as an X.509 certificate. As a result, a VSP and SP are mutually aware of their identity, thus simplifying a collaboration between these parties. In such a collaboration, a VSP could leak details about the integrity measurements to the SP. This underlines the importance of selecting a trusted VSP, that will not collaborate with an SP. In this SSO scenario, a VSP knows which SP a user accesses. Consequently, a VSP could learn about the behaviour of a user in the web. However, the communication model does not provide the VSP with an identifier for a user[19]. A VSP is therefore unable to correlate a TNC check request initiated by a specific SP to a specific user. Identifying information that might possibly be revealed in the TNCCS message exchange itself, such as licensing data of software installed on a user's machine, also does not provide a reliable way of identifying users for VSPs. Client machines can be shared by different users and users could potentially utilise more than one client machine to access SP's. If the client performs an authentication with the VSP instead of using the SAML-based SSO mechanism, the VSP is not aware of the SP's identity. In both authentication scenarios, a VSP cannot reliably track the behaviour of its users which enhances their privacy situation.

The decision whether a SAML-based SSO mechanism is used or a separate authentication at the VSP is performed is mainly based on the relation of SP and VSP(s). In a closed environment, such as an Intranet, SAML assertions can easily be used to eliminate the need for the user to perform an authentication at the VSP. In open environments, however, this approach would introduce additional administrative burden, since keys and certificates need to be managed and exchanged. This is problematic, because trust relations need to be established on an ad-hoc basis.

Analogically, the method that is used for stating the VSP requirements also depends on the underlying environment. In open environments, it is unlikely that the SP provides a list of VSPs and their X.509 certificates. The privacy gain for a user would be limited, as the SP can determine exactly which VSPs can be used. Furthermore, the administrative

[19] A similar issue exists in the Cardspace architecture, as identified in [AM07a]. However, the situation is more severe in the Cardspace architecture as it is possible to identify the user.

overhead of managing and synchronising X.509 certificates with VSPs is likely to be not bearable. Such an approach is more likely to be realised in a closed environment, in which it is predetermined which VSP is used[20]. In contrast, an approach in which the user can choose a VSP based on its capabilities is tailored for open environments. These relations are summarised in figure 4.4.

Using the mechanisms discussed in this chapter, a web-based TNC check can be performed using the proposed assertion-based attestation model. It is based on open standards that allow an easy integration into existing systems. Using a policy based approach, it provides flexibility for choosing trustworthy VSPs and integrity requirements of SPs.

However, a transport mechanism is necessary to provide a wrapping for the artefacts and the actual TNC messages. As depicted in figure 4.3, there are two separate communication paths, that is, interfaces, in the attestation-based model. Interface IF-T *(Interface Transport)* describes the communication between VSP and client for performing the integrity check. Interface *IF-PAA (Interface - Policy/Authentication/Assertion)* describes the communication between client and SP. In the next chapter, interface IF-T is further discussed, and solutions are proposed to realise IF-T in a web-based environment. The second part of the underlying communication mechanism, that is, interface IF-PAA, is discussed in chapter 6.

[20] An example of a web-based application in a closed environment is the "Verified by Visa" system, as it is managed by one organisation (Visa). The specification is available from https://partnernetwork.visa.com/vpn/global/retrieve_document.do?documentRetrievalId=119.

Chapter 5

Web-based TNC Integrity Check over IF-T

In the previous chapters, several adaptations have been discussed that allow TNC to be used in an open and web-based environment. An issue that has not been addressed so far is which underlying transport mechanism can be used to transfer artefacts between VSP, client, and SP. In this chapter, a transport mechanism is developed that facilitates communication between VSP and client. The main purpose of this communication is to transfer TNC measurement data. It is thus named like its counterpart in the original TNC architecture: *Interface Transport (IF-T)*.

As mentioned before, it is important that the chosen design can be implemented using existing open standards. This facilitates the process of integrating the TNC mechanisms into an existing authentication system. Other requirements that need to be fulfilled are discussed in section 5.1. The analysis of the requirements results in two possible approaches that can be used as a transport mechanism. The first approach is based on TLS and is discussed in section 5.2. The second approach is based on application level protocols, such as HTTP and SOAP, and further discussed in section 5.3. Finally, section 5.4 concludes this chapter and summarises the findings.

5.1 Requirements

The requirements presented in section 3.1 influenced architectural decisions made in this project and led to the proposed attestation-based model (section 3.3.3) and the related communication flow (section 4.4). Additional requirements and constraints exist that influence the choice of protocols and message formats that are considered as "candidates" for the transport mechanism. These requirements and constraints are elaborated on below.

Exchanged artefacts Several artefacts have to be transferred between VSP and client. As described in section 4.4, these artefacts are a) an authentication token, b) policies describing the TNC integrity and VSP requirements and, most importantly, c) the TNC message exchange in form of XML statements (TNCCS Batches). The user receives a

SAML assertion as a result of the integrity check. This assertion needs to be presented at the SP.

Inherited requirements IF-T, as described in the TNC specifications [Tru07g], consists of two protocols: the access protocol (e.g. EAPoL) and the AAA protocol (e.g. RADIUS)[1]. The central PDP acts as a protocol translator to translate between EAPoL and RADIUS[2]. In the assertion-based architecture, such a central protocol translator does not exist. As a consequence, both parties can use a common protocol and message format. Despite these differences, the requirements for IF-T as stated in [Tru07f, section 2.2] still apply. In summary, these requirements state that IF-T must provide an extensible and efficient half duplex message protocol, in which the integrity and confidentiality of the transferred messages are protected in order to support the use cases of the TNC architecture.

Limitation in choice of protocols The predominant protocols used by all web-based applications are HTTP[3] and its secured version HTTPS, that is, HTTP over TLS[4]. Commonly[5], HTTP connections are established using TCP port 80, while HTTPS connections use port 443. Communication on other ports is sometimes blocked in companies and other organisations. In order to prevent TNC traffic from being blocked by firewalls or other mechanisms, such as proxy servers, only those mechanism that are based on HTTP or TLS as the base mechanism are considered. This excludes any protocols that would use a custom TCP-based message exchange and other application level protocols.

Open standards Furthermore, using standard authentication protocols and message formats is the preferred option rather than developing new protocols. To accomplish the goals of TNC in a web-based environment, extension mechanisms of existing mechanism and protocols can be used. By using this approach, a web-based TNC solution can benefit from the experience with existing technologies and avoid pitfalls that occur with newly designed protocols and protocol extensions.

In the following sections, protocol and message format "candidates" are discussed. The next section discusses TLS based approaches, while application level based approaches are covered in section 5.3.

[1] Cf. [Tru07f, figure 2].
[2] Cf. figure 2.4 on page 26.
[3] The current version 1.1 of the HyperText Transfer Protocol is defined in RFC 2616 [Net99b].
[4] RFC 2818 - HTTP Over TLS [Net00] gives an overview about how these protocols are combined.
[5] As described in the list of *well known ports* provided by the IANA at http://www.iana.org/assignments/port-numbers.

5.2 Analysis of an SSL/TLS Approach

HTTP traffic in security sensitive web applications can be protected at the transport layer using SSL/TLS[6]. This combination of HTTP and TLS, known as HTTPS, provides confidentiality and integrity protection as well as support for authentication.

In a HTTPS connection, the HTTP protocol is tunnelled within a secure channel that is established by the *TLS Handshake Protocol*. Before this channel is established mutual authentication can be performed as part of the handshake protocol. The application data (in this case HTTP requests and responses)[7] is transferred according to the *TLS Record Protocol* which provides confidentiality and integrity protection [For07]. As the TNC integrity check shall be implemented as part of an existing authentication mechanism, the following discussion focusses on the handshake protocol, which is responsible for authentication in TLS.

The idea of using TLS for integrity checks has been proposed previously. [Tru07b] briefly proposes the creation of a dedicated TLS extension that is capable of performing the TNC integrity check in web-based systems. However, no attempt is made in [Tru07b] to define such an extension. As stated previously, this thesis does not aim at creating a customised protocol extension and an approach that is based on existing mechanisms is preferred instead. Another TLS based approach is proposed in [GSS+07]. This TPM-based system uses the SKAE[8] X.509 certificate extension to perform a binary remote attestation during the TLS handshake. Unlike binary remote attestation, TNC requires multiple round-trips to finish an integrity check. However, the proposed SKAE-based approach is not capable of performing multiple round-trips and thus cannot be used for TNC integrity checks.

The number of round-trips in a standard TLS handshake is also fixed. More importantly, TLS does not offer the possibility of exchanging additional information (such as the artefacts identified in section 5.1) in a semantically encapsulated manner during the handshake. However, the TLS handshake protocol can be extended as described in RFC 4366 ([Bla06]). This extension mechanism, which is described in the next section, is the base for existing TLS protocol extensions that are considered in sections 5.2.2 et seq. for encapsulating the TNC message exchange.

[6] TLS 1.1 is defined in RFC 4346 [Net06a]. TLS 1.0 is the successor of the SSL 3.0 protocol originally developed by Netscape. TLS 1.0 is therefore sometimes referred to as SSL3.1 [Net99a]. For the remainder of this document SSL/TLS is substituted with TLS unless a distinction is necessary.

[7] While not within the scope of this document, in addition to HTTP traffic, TLS can also be used for securing access to emails (using IMAP, SMTP, or POP3) and VPNs.

[8] Subject Key Attestation Evidence as defined in [Tru05]. This technique uses the extension mechanism in X.509 Public Key Certificates to store additional information.

5.2.1 TLS Handshake and its Extension Mechanism

Before a secure connection can be established in TLS, security parameters need to be agreed on, certificates exchanged, and an encryption key derived. The TLS handshake protocol is responsible for performing the aforementioned tasks. Along with its extensions mechanism it is depicted in figure 5.1 and described in the following.

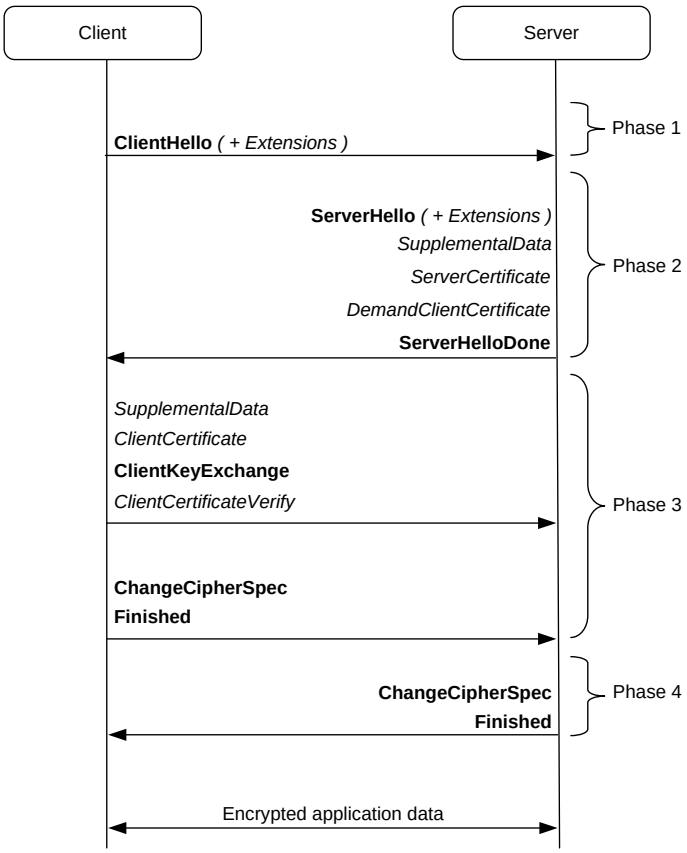


Figure 5.1: TLS Handshake Protocol with Extension Mechanism. Bold message names indicate mandatory messages, while italic names refer to optional messages.

TLS is not limited to a specific set of cryptographic mechanisms. Instead, TLS defines a flexible mechanism that allows the communicating parties to negotiate a cryptographic algorithm during the handshake. Before a secure connection can be established, it is thus necessary that the participating parties agree on security parameters (such as the mutually available TLS protocol version, supported cipher suite, and compression algorithm). This negotiating process is encapsulated in two *Hello* messages, as described below [For07, Bla06].

Phase One The first message, *ClientHello*, initiates the TLS connection establishment. This message carries parameters (amongst others) that indicate cipher and compression algo-

rithms supported by the client. In addition, a random value is transferred that is used later in the protocol for deriving an encryption key.

The client can indicate its support for TLS extensions using the extension mechanism defined in [Net06b]. An extension is associated with an extension number. The list of all standardised extensions and their numbers are maintained in a public list[9]. Depending on the extension that should be used, the protocol allows for the inclusion of additional data in two places during the messages exchange. Firstly, data can be placed in the ClientHello message along with the extension number. In addition, data can be transferred between client and server during the handshake protocol(in phases 2 and 3) as depicted in figure 5.1 as *SupplementalData*. If a TLS connection already exists, the ClientHello message initiates a renegotiating of the security parameters.

Phase Two The server chooses a TLS version, a cipher algorithm, and a compression algorithm from the list provided by the client. These values are placed into the *ServerHello* message to confirm the selection. In addition, the server can also indicate its support for extensions by including the corresponding extension numbers into the *ServerHello* message. If the server supports extensions that have been previously proposed by the client, extension specific data can now be exchanged.

Authentication in TLS is (usually) realized using X.509v3 identity certificates that contain a public RSA key[10]. Such a certificate is attached to the *Certificate* message. This gives the client the possibility of validating the server's identity by comparing the domain name contained in the certificate with the URL used to connect to the server.

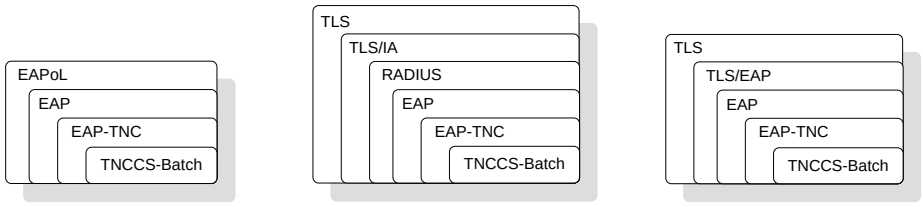
TLS also allows client authentication. This mechanism, which is typically based on certificates, can be triggered by the server using a *CertificateRequest* message. This message can include certification authorities that are accepted by the server. To indicate the end of the second phase, the server sends a *ServerHelloDone* message.

Phase Three At this stage of the protocol, the client has the chance to include extension specific data into the handshake protocol (indicated as SupplementalData in figure 5.1).

If the client has provided a certificate, then a proof of ownership is sent in the *CertificateVerify* message that includes a digital signature of the previous message exchange created using the client's private key.

[9] This list is available at http://www.iana.org/assignments/tls-extensiontype-values and is maintained by the Internet Assigned Numbers Authority. In order to define a new extension type, an application needs to be made to IANA through the IETF Consensus process [Bla06].

[10] The TLS specifications also define other mechanisms, such as a Diffie-Hellmann-based key exchange ([DH76]). The following discussion is limited to RSA-based certificates, as they are most commonly used. In addition, TLS extensions exist that do not rely on certificates at all. However, these extensions either rely on a shared secret (e.g. a pre-shared key as in [ET05]) or need additional hardware (e.g. a special one-time-password generator such as in [OHB06, OHB+07]).



(a) TNC over EAPoL. (b) TNC over TLS/IA. (c) TNC over TLS/EAP.

Figure 5.2: Comparison of three protocol stacks: EAPoL, TLS/IA, and TLS/EAP for transporting TNCCS-Batch messages

In the next step, the client generates the *pre-master key*. This key is used to calculate the encryption key that is later used to protect the application data. Using the *ClientKeyExchange* message, the pre-master key (encrypted with the server's public RSA key) is transferred to the server. Server and client can now each compute the master key by using the pre-master key and the random values exchanged in the ClientHello and ServerHello messages.

After the client has computed the master key, it informs the server, using the *ChangeCipherSpec* message, that all further communication will be encrypted using the cipher suite and key previously agreed on.

Finally, the client ends the handshake protocol by sending an encrypted *Finished* message that contains a cryptographic hash of the previous exchanged messages.

Phase Four The server decrypts the *Finished* message received from the client and verifies the hash value. After a successful validation, the server sends a *ChangeCipherSpec* and *Finished* message to the client. The handshake protocol is now completed and encrypted application data can be exchanged between client and server.

The mechanism described above provides a possible means of exchanging additional information during the TLS handshake. As described in the introduction to this chapter, the goal of this thesis is not to define a completely new protocol and message format on top of TLS. Instead, existing protocols shall be used to perform the TNC check.

A number of TLS extensions are defined which extend the TLS handshake for different purposes. The extensions defined in RFC 3546 ([Net06b]) focus merely on memory limited clients. Examples include negotiation of fragment length and sending URLs that point to certificates instead of sending the certificate itself[11].

Further extensions have been proposed outside of RFC 3546. Those that can potentially be used to realise a TLS-based TNC integrity check using the assertion-based architecture are discussed in the following sections.

[11] In order to allow a memory limited client to store only the private key and URL instead of a whole certificate.

In the TNC specifications [Tru07f], IF-T defines the interface between Network Access Requestor (that is, the client) and the Network Access Authority (that is, the VSP). As described in section 2.3, TNC in local area networks uses EAP over LAN (EAPoL) to exchange integrity check measurements. The actual TNC messages are encapsulated into an EAP authentication method called EAP-TNC[12]. Figure 5.2a[13] visualises this encapsulation.

EAP is widely used as an authentication protocol in local area networks. Some work has already been done to integrate EAP into the TLS protocol. This approach wraps the EAP-based protocol exchange into TLS, using the TLS handshake protocol as the transport mechanism. The following sections discuss whether these approaches can be used as the transport mechanism for TNC integrity checks.

5.2.2 TLS/IA - TLS Inner Application Extension

The *TLS Inner Application Extension (TLS/IA)*[14] defines a mechanism to transport RADIUS messages over TLS. More precisely, it allows the transfer of attribute-value pairs defined in the RADIUS/Diameter namespace during the handshake protocol. As depicted in figure 5.3[15], the RADIUS messages are exchanged after the ChangeCipherSpec message has been sent (they are therefore encrypted) and before application data is exchanged. As RADIUS is capable of transporting EAP messages[16], TLS/IA can also transfer EAP encapsulated TNC integrity messages. However, as figure 5.2b suggests, the additional RADIUS layer in TLS/IA produces a complex protocol stack. The RADIUS layer does not provide additional functionality for the TNC use case and is therefore not necessary. An approach that does not require this layer is described in the following section.

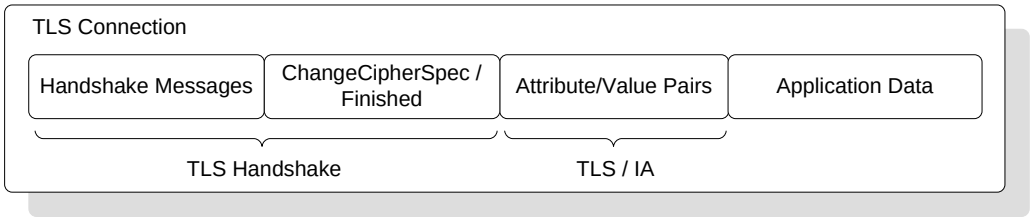


Figure 5.3: Schematic overview of TLS protocol with TLS/IA

[12] As described in section 2.3.4.
[13] This figure is based on figure 3 in [Tru07f].
[14] This Internet draft is specified in [TLS06].
[15] Based on the figure "In TLS/IA" in [Fun03].
[16] As described in section 2.3.

5.2.3 TEE - TLS EAP Extension

A solution that avoids the RADIUS layer is provided by the *TLS EAP Extension (TEE)*[17]. This extension has been proposed as a less complex alternative to TLS/IA. TEE defines a mechanism to transport EAP packages directly over TLS. Figure 5.2c summarises the resulting protocol stack of TEE for transporting TNC messages. The primary use case of TEE is to allow legacy applications to perform EAP-based authentication using TLS, instead of using application level protocols.

The EAP authentication method EAP-TNC is used for TNC in local area networks. This authentication method can be incorporated into a TEE message exchange, as described in the following. Figure 5.4[18] shows the phases of a TLS handshake and the TEE messages that are incorporated into the handshake protocol.

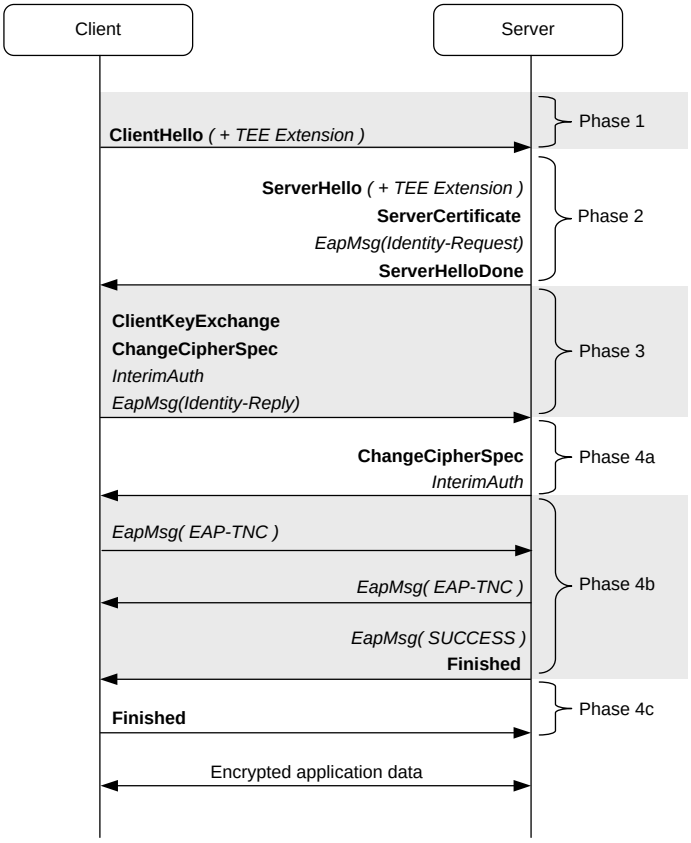


Figure 5.4: Integrating EAP encapsulated TNC messages into TEE. Bold message names indicate TLS handshake messages while italic message names represent TEE messages.

To indicate support for TEE, both the client and the server include the TEE extension type in their *Hello* message. EAP messages are encapsulated in a new message type called *EapMsg*. During phase 2, the server can use this message type to request identity information from

[17] An Internet draft proposed in [TLS07].
[18] Based on [TLS07, section 3].

the client. For server authentication, existing methods such as X.509 identity certificates can be used.

Instead of sending a *Finished* message in phase 3, the client sends an *InterimAuth* message. This newly defined message type is syntactically equivalent to the *Finished* message but has semantic differences. While the *Finished* message indicates that application data can now be sent, the *InterimAuth* message announces that further EAP messages can be exchanged. Because the *ChangeCipherSpec* message has already been sent, all message exchanges are encrypted using the cipher suite agreed on during phase 1 and 2. In the example message exchange in figure 5.4, the client performs its authentication by sending an EAP *Identity-Reply* message.

Phase 4, as described in section 5.2.1, is split into three phases: 4a, 4b, and 4c. In phase 4a the server announces that all upcoming messages are encrypted. Further EapMsg messages can be exchanged in phase 4b. The example in figure 5.4 shows EAP-TNC embedded in an EapMsg message[19]. In such an approach, a TLS server needs to extract and forward TNC messages to the TNC Server, which is either co-located with the TLS server or realised as a separate application or server. Assuming a successful TNC integrity check outcome, the server can send an *EAP-Success* message indicating a successful authentication to the client. Phase 4b is finalised by sending a *TLS Finished* message, indicating that application data can now be exchanged. This is acknowledged by the client in phase 4c.

5.2.4 Summary and Verification of Suitability

Using a TLS-based approach for the TNC integrity check is a promising approach as TLS is widely used to protect message exchanges at the transport layer. It can be used for a variety of upper layer protocols, such as HTTP and POP3, and it is well tested. To incorporate the additional requirements of TNC, it is necessary to use extension mechanisms on top of TLS. Two extension mechanisms have been analysed in this chapter.

TLS/IA and TEE provide a way to exchange EAP messages during the TLS handshake. Both protocol extensions allow legacy applications to use EAP-based authentication at the TLS layer, that is, without changing the application layer. Because TEE is less complex than TLS/IA, TEE is the preferred method for including TNC in a TLS handshake. The TEE approach enables EAP authentication to be decoupled from EAPoL and to be integrated into the TLS handshake. This protocol extension is a promising approach for realising the simpler direct and relayed attestation model[20], as these models do not require additional artefacts to be exchanged. However, issues exist when considering TEE for the assertion-based attestation model.

[19] Cf. section 2.3.4 and figure 2.4 for a description of EAP-TNC.
[20] Cf. sections 3.3 and 3.3.2.

As discussed in sections 3.3.3 and 4.1, additional artefacts (SAML assertions and policies) need to be transferred in addition to the TNC messages (TNCCS-Batches). However, the TEE mechanism does not provide a method for including additional information. It is hence not possible to include these artefacts inside a TEE message. Another possible means of including these artefacts in the message exchange would be at the TNCCS-Batch level. However, according to the TNCCS-Batch specifications ([Tru07g, chapter 3.3]) no extension mechanisms exist that would allow this. Including additional artefacts would thus violate the current TNC specifications.

In summary, TLS-based protocol extensions cannot be used to fully satisfy the requirements stated in section 5.1. This approach does not allow for the inclusion of additional artefacts in a way that conforms to existing standards. Incidentally, using EAP as a wrapper around TNC messages adds a layer of complexity to the web-based TNC check while at the same time providing very little benefits. The only advantage of keeping the outer EAP layer is that existing implementations of the LAN-based protocol stack can be partially reused.

A general drawback of implementing TNC over TLS is that the underlying TLS infrastructure needs to be changed. TLS support is not only implemented in software (for example in the Apache Web server module *mod-ssl*[21]), but can also be found in hardware-based TLS accelerators that perform the TLS handshake on behalf of a Web server. This configuration is deployed to avoid performance bottlenecks on the Web server created by TLS connection establishments. If such a configuration is used, the Web server is unaware of the TLS session as the TLS tunnel is terminated at the TLS accelerator. In such a deployment, it would be the TLS accelerating hardware's responsibility to either perform the TNC check or to forward TNC related messages to the TNC application. An approach that avoids these implementation issues is to perform the TNC integrity check directly at the application level. This approach is discussed in the following sections.

5.3 Analysis of an Application-level Approach

As discussed in section 5.2, the approach of using TLS as the underlying protocol has some limitations. In this section it is analysed whether the limitations can be overcome by using a protocol at a different level. From a network point of view, the lowest application protocol level of web-based applications is the HTTP protocol.

Section 5.3.1 briefly introduces the principle mechanism of HTTP. SOAP, which is based on HTTP, and its applicability for TNC are discussed in section 5.3.2. SOAP itself is only a container for XML based messages. Several message formats have been defined to unify SOAP-based communication. These message formats enable an automatic message exchange in a standardised manner. Instead of defining a proprietary and TNC-specific message format,

[21] See http://www.modssl.org/.

existing formats shall be leveraged to the requirements of TNC. The properties of two existing approaches, SAML protocols and WS-Trust, are analysed in sections 5.3.3 and 5.3.4, respectively. Finally, section 5.4 summarises the results of these analyses.

5.3.1 The Base Protocol: HTTP

HTTP is the common denominator of all web-based applications. HTTP, as defined in RFC 2616 [Net99b], is a request/response driven and text-based protocol. A HTTP request consists of three blocks, outlined in the following. Listing 5.1 provides an accompanying example HTTP request.

Request-Line The request-line consists of a request method, a request URI, and the HTTP version that is used (cf. line 1). HTTP supports a variety of request methods for accessing a resource. The most common methods are *GET* and *POST*. The *GET* method is a *safe method*, that is, it is used to retrieve a resource without changing a state on the server. *POST* method can be used to append additional data to a request. For example, data that is entered and submitted in an HTML form[22] is usually transferred using the HTTP POST method.

Headers HTTP defines a large number of headers for various purposes. Examples include cache control header, content type (e.g. *application/x-www-form-urlencoded* for an HTML form, cf. line 3), and document size.

Message Body The header section is separated from the message body using the character sequence CR LF (carriage return & line feed). The message body is optional and typically used for POST requests, where it contains additional data that supplements the request (cf. line 4).

```
1 POST /application/formHandler.ext HTTP/1.1
2 HOST: www.example.com
3 Content-Type: application/x-www-form-urlencoded
4 username=myUsername\&email=me@myTLD.com
```

Listing 5.1: An example HTTP POST request and response

After an HTTP request has been processed by an HTTP server, it generates an HTTP response. The scheme of a HTTP response is similar to an HTTP request and is summarised in the following. Listing 5.2 gives an example.

Status-Line The status-line includes the HTTP version, a status code, and a reason phrase. The status code and reason phrase indicate the status of a request (cf. line 1). For

[22] The *Hypertext Markup Language (HTML)* is a language used to describe documents that are typically provided by a Web server and displayed in a Web browser. HTML forms provide a way for users to send information back to the Web server [TS06, page 547 et seqq.].

example, "200 OK" indicates that a request was successful, while "403 Forbidden" indicates that a given resource cannot be accessed due to access restrictions.

Headers Like HTTP requests, HTTP response headers are defined for a wide range of applications, and provide meta information about the message body. Examples include the content type (e.g. *text/xml* for an XML document) or its size (cf. line 5).

Message Body The message body contains the requested resource, for example, the HTML representation of a Web site (cf. line 9).

```
1 HTTP/1.1 200 OK
2 Date: Mon, 29 Feb 2008 22:38:34 GMT
3 Server: Apache/1.3.3.7 (Unix)  (Red-Hat/Linux)
4 Last-Modified: Thu, 08 Feb 2007 23:11:55 GMT
5 Content-Length: 438
6 Connection: close
7 Content-Type: text/html; charset=UTF-8
8
9 <html>....</html>
```

Listing 5.2: An example HTTP response.

HTTP can be used to transport binary and text-based information. The message exchange during a TNC check is XML-based, making it possible to transfer all artefacts within the HTTP message body. However, transferring plain XML documents over HTTP[23] lacks a standardised frame. As a result, there are no mechanisms for fine grained error handling (other than the standard HTTP status codes), and no standard handling for message level security. SOAP has been developed to give XML messages such a frame, as further elaborated on in the following section.

5.3.2 Transferring XML Messages with SOAP

The current version (v1.2) of the SOAP specification was published as a W3C[24] recommendation[25]. SOAP is an XML-based packaging scheme that was created to transport XML data between applications using standardised transport mechanisms. A generic framework is defined in the SOAP specifications that describes how arbitrary transport mechanisms can be used to transport SOAP. However, in practice HTTP is is the most commonly used transport protocol [RR04, pages 7 and 49 eqq.][Mit07, section 2.2.2].

[23] Sometimes referred to as "plain old XML", or POX for short [LKS06].

[24] The World Wide Web Consortium (W3C) is a standards organisation publishing web related standards. Examples of standards published by the W3C include HTML, XML, and RSS. More information can be found at http://www.w3.org.

[25] Published in June 2003 in [Gud03] and updated in April 2007 [GHM+07]. Until version 1.1 of the SOAP specifications, SOAP used to be an abbreviation for *Simple Object Access Protocol*. However, this abbreviation was dropped as it does not reflect the intentions behind SOAP, that is, SOAP's purpose was and is not to access objects [RR04, pages 51 eq.].

The SOAP message exchange supports two different encoding styles for transferring XML message: *Remote Procedure Calls (RPC)* and *Conversational Message Exchanges*. In earlier versions of SOAP, RPC-based message exchange was emphasised as the universal solution for request/response based communication. However, in the current version of SOAP, RPC-based communication is only used when programmatic behaviour shall be reflected within SOAP. That is, platform independent function calls are made to a remote host using SOAP as a middleware[26].

The second message encoding, that is, conversational message exchanges, is the preferred method when XML-based content shall be exchanged in a "conversation"-like manner, that is, using a request/response pattern [Mit07, section 2.2]. This conversation pattern also occurs during a TNC message exchange, in which TNCCS Batches are sent back-and-forth during a TNC measurement. This SOAP mode is therefore applicable for the TNC message exchange.

The *SOAP Request-Response Message Exchange Pattern* defines how a conversation-based message exchange is realised using HTTP [Gud07]. A SOAP request is sent using the HTTP POST method with the content type[27] *application/soap+xml*. The content type header indicates to the receiving Web server how the message content needs to be handled.

The structure of a SOAP message consists of three elements: *Envelope*, *Header*, and *Body*. A schematic overview is depicted in figure 6.3 (a). Figure 6.3 (b) shows a simplified version of the equivalent XML representation of a sample SOAP message. As SOAP is an XML-based format, it requires a single root element, the *SOAP Envelope* [Bra06, Section 2.2].

A SOAP message can have an arbitrary number of headers defined in the *SOAP Header* section. Headers are the extension mechanism in SOAP, intended to add features and functionality to SOAP messages independent of the application payload. An important SOAP header is defined by the *WS-Security (WSS)* specification [NKMHB06]. WSS defines methods for adding message security to SOAP messages, that is, they describe how to authenticate, encrypt, and sign SOAP messages. WSS uses the *security* header to add message-based security in a standardised manner. In the example in figure 6.3 (b), a SAML token is included to authenticate a SOAP request[28].

Finally, the mandatory SOAP body contains the application payload. In the example in figure 6.3 (b), the SOAP body contains a TNCCS Batch. With SOAP faults being the exception, the content of the SOAP body is not specified and are application dependant.

[26] Middleware provides a common programming abstraction across distributed systems designed to hide complexity. It is defined as a layer of software located between the operating system and an application. When the RPC encoding is used, function calls, parameters, and return values are translated into XML using standardised serialisation and de-serialisation mechanisms [RR04, page 51][Bak03].

[27] Cf. section 5.3.1.

[28] Cf. section 4.3.2.

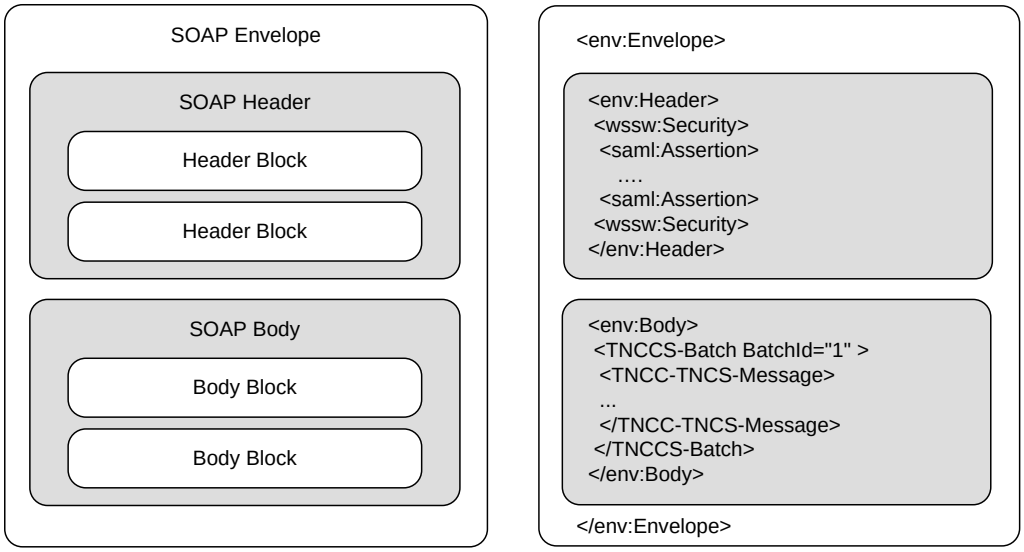


Figure 5.5: SOAP message structure (a) and simplified XML representation of a SOAP message (b)

SOAP faults are a mechanism to express errors that occurred while processing a SOAP message. Such an error can arise from a number of causes, from application-specific (e.g. TNC policy file not valid) to issues with the SOAP message itself (e.g. header not supported). The difference between the SOAP fault mechanism and other error reporting schemes[29] is that it allows the inclusion of fine-grained details about the error. For instance, in addition to error codes, SOAP allows the inclusion of application specific error details (e.g. in XML) and error messages for end users in multiple languages.

In summary, SOAP provides a standardised way of transporting XML-based messages using various transport mechanisms. For the TNC message exchange, HTTP is the most important transport mechanism. SOAP defines a fault mechanism which can be used to express errors in a verbose way. However, like HTTP, SOAP intentionally lacks a semantic wrapper for the payload. That is, it is not possible to encapsulate data in a meaningful way only relying on SOAP. A semantically meaningful encapsulation requires a message format. Instead of defining a new message format, an existing format shall be leveraged for encapsulating TNC check messages. In the following, standardised message formats are evaluated.

5.3.3 Using SAML protocols for IF-T

SAML assertions have been introduced as a mechanism to encapsulate a TNC check result in section 4.3.3. These assertions are part of the SAML architecture, that contains several

[29]Such as HTTP status codes. Cf. section 5.3.1.

additional concepts. These concepts can be combined to realise specific use cases, such as a Single-Sign-On (SSO) system[30].

As part of such a use case, the SAML specification defines how SAML assertions can be requested using SOAP as the transport mechanism. In the following, a brief overview is given, outlining the principal concepts of this mechanism. Furthermore, this section analyses whether this mechanism can be used as the underlying transport mechanism for TNC integrity checks.

The basic SAML 1.1 architecture [HM04][31], consists of four components.

Assertions can carry authentication, attribute, and authorisation information.

Protocols define message formats for requesting and returning assertions.

Bindings define how SAML protocols map onto standard messaging or communication protocols, such as SOAP.

Profiles describe how SAML protocols, bindings, and assertions can be combined and constrained to support a defined use case.

The relationship between these concepts is shown in figure 5.6. SAML Assertions are packaged in protocols. Using a binding, these protocols are mapped to an underlying transport mechanism. Profiles combine a set of assertions, protocols, and bindings to form the basis of a use case.

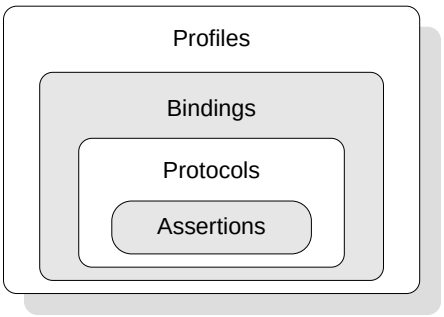


Figure 5.6: Basic SAML architecture showing the relationship of SAML concepts

With regard to finding a transport protocol for the TNC integrity check, the SAML concepts of protocols and bindings are of interest. A SAML protocol describes a message format for requesting and returning SAML assertions [MMP03, section 3] [SC05, section 3]. Several protocols are defined for requesting different types of SAML assertions. Attribute

[30] Cf. section 4.3.2.

[31] SAML 2.0 defines two additional concepts (*Metadata* and *Authentication Context* [SAM08, section 4]) which are not relevant here and have therefore been omitted for brevity.

assertions, which are used to encapsulate the TNC check result[32], are requested by using *Attribute Queries* [MMP03, section 3.3.4][33].

These request and response messages are mapped to standard messaging or communication protocols using SAML bindings. An example is the SAML SOAP binding[34]. It describes how SAML request and response messages are exchanged using the SOAP message format. Figures 5.7 and 5.8[35] show an example message exchange and a corresponding schematic illustration thereof, in which an attribute query is used to request an assertion using a SOAP (over HTTP) binding.

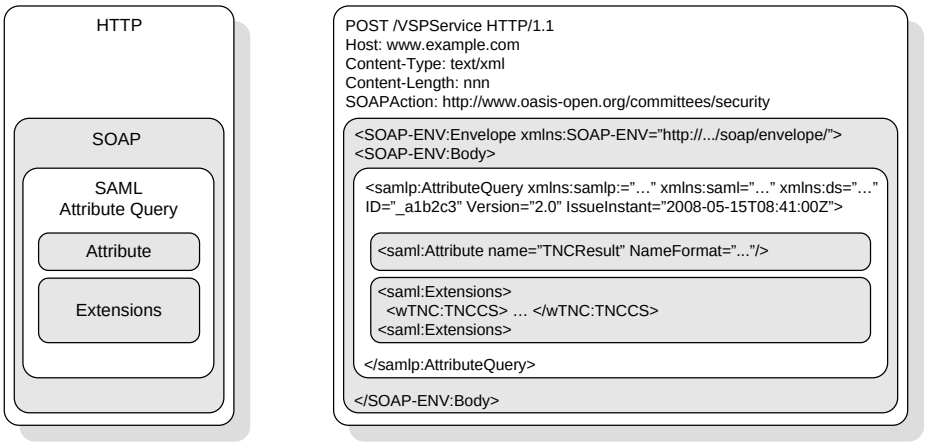


Figure 5.7: Using the SAML protocol for requesting a SAML assertion using an Attribute Query over SOAP and HTTP

Figure 5.7 shows a SAML attribute query embedded in a SOAP message. This example message shows how clients can initiate the TNC check with a VSP. The attribute query contains two elements. The *Attribute* element states which attributes are requested, that is, which attributes are expected to be contained in the returned assertion. In the example, *wTNC:TNCResult* requests the result of a web-based TNC integrity check. For the TNC integrity check, additional messages, such as TNCCS-Batches, need to be transferred[36] between a client and a VSP. SAML 2.0 introduces an extension mechanism that allows the inclusion of additional information in SAML request and response messages. This extension mechanism does not exist in previous versions of SAML, thus preventing them from being used for the TNC message exchange. In SAML 2.0, additional information can be attached to an extension point, that is, the element *Extensions*. In the example in figure 5.7, a TNCCS-Batch is attached to this element.

[32] Cf. chapter 4.
[33] In SAML 2.0, Attribute Queries are part of the *Assertion Query and Request Protocol* [SC05, section 3.3.2.3].
[34] This is the only binding supported by SAML 1.1 [SAM03, section 3.1]. SAML 2.0 defines additional bindings that are based on SOAP, HTTP, and variations thereof [SAM05, section 3].
[35] Both figures are based on [SAM05, section 3.2.3.5] and adapted for the TNC use case.
[36] Cf. section 5.1.

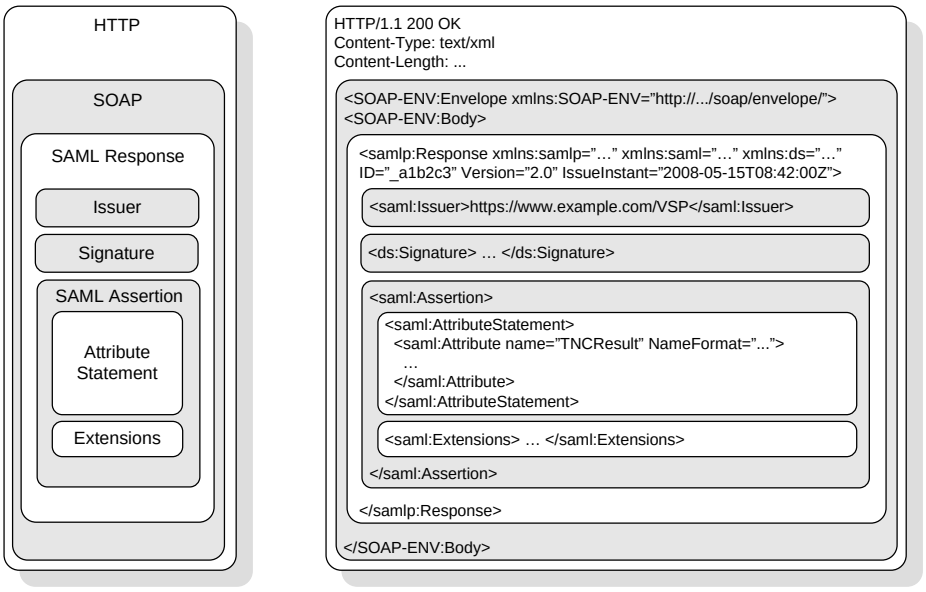


Figure 5.8: Returning a SAML assertion using the SAML protocol over SOAP and HTTP

After the request has been processed, a response is generated (see figure 5.8) that contains the requested attribute assertion in the *Response* element. The assertion contains the requested attribute, that is, a TNC check result. Additional TNC related information (such as TNCCS-Batches) can be attached to the *Extensions* element. Furthermore, the *Response* element contains the issuer of the assertion and a digital signature to authenticate the message and protect its integrity, as described in section 4.3.2.2.

As described in section 5.1, a TNC check can require several round trips before an access recommendation can be made. The SAML protocol, however, only provides a simple request/response mechanism. To overcome this limitation, several request/response message pairs could be strung together. However, two restrictions in the SAML protocol prohibit such an approach [SAM05, section 3.2.2.1]. A SAML response must include either a SAML assertion or a fault message. During the TNC message exchange, it is not possible to issue an assertion until the check is finished[37]. The only alternative to sending an assertion is to send a SOAP fault message instead. Such a fault message could indicate that the TNC check is not completed yet, as additional information needs to be obtained from the client. However, the SAML protocol does not foresee a mechanism that would allow a client to respond to such a response. Instead, [SAM05, section 3.2.2.1] explicitly prohibits sending a reply to a SAML response message. The lack of an extensible request/response mechanism prevents the SAML protocol being used as the transport mechanism for a TNC check. In the following section, a mechanism is described that does not have this limitation.

[37] It would be possible to issue empty assertions. However, in such an approach, several SAML assertions need to be created, signed, and transmitted unnecessarily, as they do not carry any information.

5.3.4 Using WS-Trust for IF-T

The SAML request/response mechanism, as discussed previously, is not flexible enough to fulfil the requirements for realising a web-based TNC integrity check. A mechanism that does not have similar limitations is WS-Trust [Nad07], which will be discussed in the following.

WS-Trust is a WS-* specification and an OASIS standard that provides extensions to WS-Security (WSS). More specific, WS-Trust defines mechanisms to deal with the issue and exchange of security tokens.

Authentication in WSS is provided by including a security token in the header of a SOAP request. Examples of security token types include username and SAML tokens. Before a SOAP request is processed, the attached security tokens are validated. It follows that authentication in WSS works the same way as in other authentication systems that validate a user's identity and authorisation before a request is processed. Before a client can use a security token, the token has to be obtained. This process is straightforward for a username token, as this token can be created by a user. However, a service provider might require a SAML token issued by a special type of third party, called a *Security Token Service (STS)*. In such a situation, a user needs to contact and obtain a token from an STS before the service can be used.

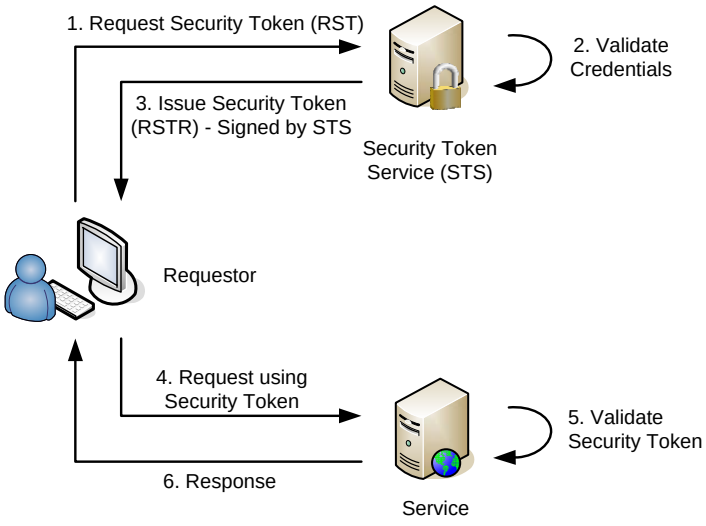


Figure 5.9: Obtaining a security token using WS-Trust

WS-Trust defines a mechanism to achieve this. An example that shows how WS-Trust can be used is depicted in figure 5.9. A user (called a *Requestor* in WS-Trust) seeks access to a service. However, the Web service requires the requestor to present a special token before access to the service is granted. In step 1 the requestor asks for a security token to be issued by the STS by sending a *RequestSecurityToken (RST)* message. Typically, a requestor has to

provide an authentication token in order to receive back the requested token. For example, a username token can be exchanged for a SAML token that includes assertions about the requestor. If the presented token validates successfully (step 2), the STS issues the requested token (step 3) in a message of type *RequestSecurityTokenResponse (RSTR)*. The requestor can use the obtained token to access the service in step 4. The token is validated (step 5) and a response is sent to the requestor (step 6).

When comparing the WS-Trust message exchange above with the proposed message flow in the assertion-based attestation model, several similarities become apparent. In both models a security token needs to be obtained and presented before a service can be used. Furthermore, the client plays a central role in both models. Because of these similarities, WS-Trust could potentially be used to request a security token that contains the TNC check result. Rather than only validating the credentials, an STS would act as a VSP and also validate the integrity state of a client. In such an approach WS-Trust is used to encapsulate the TNC message exchange and the token issue. Before discussing if and how WS-Trust can be leveraged to encapsulate the TNC integrity check in section 5.3.4.2, the next section briefly describes the message formats defined by WS-Trust.

5.3.4.1 WS-Trust Message Formats

WS-Trust is a request/response-based protocol. Requests for security tokens are encapsulated in *Request Security Token (RST)* messages. A token is returned in a *Request Security Token Response (RSTR)* message. In the following, a brief overview of the important aspects of the message format is given.

A sample RST message is shown in listing 5.3. In the following, a short description of each attribute and element is given[38].

```
<wst:RequestSecurityToken xmlns:wst="...">
 <wst:TokenType>... </wst:TokenType>
 <wst:RequestType>... </wst:RequestType>
 <wsp:AppliesTo>... </wsp:AppliesTo>
 ...
</wst:RequestSecurityToken>
```

Listing 5.3: A simple RST message frame

As mentioned earlier, WS-Trust supports several types of token. A requestor can indicate the required token type using the */wst:RequestSecurityToken/wst:TokenType* element. Token types are usually referred to using a URI. WS-Trust is not restricted to issuing tokens, but can also be used to renew or validate tokens. Using the */wst:RequestSecurityToken/*

[38] The description below utilises the same scheme that is used in the WS-Trust specification for referring to an element or attribute. This scheme uses the notion of a path to address an element. Attributes are addressed using the "@" character.

wst:RequestType element, it is possible to indicate what specific action shall be performed. These actions are encoded in URIs[39].

Instead of explicitly stating a specific token type, it is also possible to indicate the scope of a requested token using the */wst:RequestSecurityToken/wsp:AppliesTo* element. An example of the scope of a token is the URL of an SP. An STS is then required to have previous knowledge about which token type is required for a given scope.

RST messages are extensible, that is, additional information can be passed to an STS in a token request (*/wst:RequestSecurityToken/any*). If an STS does not understand a particular extension element, it is expected to return a SOAP fault instead of an RSTR.

An STS responds to an RST with an RSTR message. This message usually includes the requested token (or a SOAP fault in case an error has occurred). Listing 5.4 shows an example RSTR message. A description of the contained elements is given below.

```
<wst:RequestSecurityTokenResponse xmlns:wst="...">
 <wst:TokenType>... </wst:TokenType>
 <wst:RequestedSecurityToken>... </wst:RequestedSecurityToken>
 ...
</wst:RequestSecurityTokenResponse>
```

Listing 5.4: A simple RSTR message frame

The type of the requested token is returned in the */wst:RequestSecurityTokenResponse/wst:TokenType* element. The issued token itself is placed into the */wst:RequestSecurityTokenResponse/wst:RequestedSecurityToken* element. Alternatively, it is also possible to provide a URI that points to the issued token. An RSTR message supports the same extension mechanism (*/wst:RequestSecurityTokenResponse/any*) as an RST message, that is, additional information can be appended.

In this section the message format of WS-Trust has been outlined. The next section discusses how this message format can be used for a web-based TNC message exchange.

5.3.4.2 Extending WS-Trust for IF-T

WS-Trust defines a mechanism to issue security tokens. In many cases, the issuance of a token requires that the requestor supplies another token, such as a username token to prove his or her identity. In the result, a token is exchanged for another token. As mentioned earlier, this protocol flow has similarities to the protocol flow in the assertion-based attestation model. However, instead of only relying on a token that is presented by the requestor, an integral part of the token issuance is the TNC integrity check. Thus, issuing the token relies on an optional authentication and the integrity state of a client.

[39] For example, the URI "http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue" indicates that a token is requested.

RST and RSTR messages are used to request and issue tokens. In addition to token specific information, such as the requested token type, WS-Trust defines an extension mechanism that allows arbitrary XML-based data to be included in RST and RSTR messages, as described in the previous section. This extension mechanism can be used to include the TNCCS-Batches that need to be exchanged between client and VSP. That is, the TNC message exchange is transferred using WS-Trust. The result of a TNC integrity check, encapsulated in a SAML token, is returned to a client in an RSTR message.

As described earlier, TNC requires multiple round-trips before an access recommendation can be made. WS-Trust, as described so far, can be used to request a security token using a simple request/response protocol, that is, WS-Trust does not support multiple round-trips. However, an extension mechanism that provides such capabilities is described in the WS-Trust specification: the *negotiation and challenge extension*. The WS-Trust specification describes an authentication of a requestor based on certificates as a primary use case for this extension. Before an STS issues a token, it *challenges* the requestor to sign a random value. The signature is then sent to the STS where it is validated in order to ensure message freshness [Nad07, section 8.6]. However, this extension mechanism has been specified so that it can be used to exchange additional messages between the initial RST message and the message that contains the requested token. By combining this mechanism with the extension mechanism described in section 5.3.4.1, it is thus possible to exchange arbitrary information in multiple round-trips before a token is returned. This combination of extension mechanisms can be exploited to encapsulate the TNC integrity check, as shown in figure 5.10 and described below.

1. As described in section 5.3.4, the first message in a WS-Trust message exchange is a *Request Security Token (RST)* message. This message is sent from a requestor to an STS/VSP to request a specific token. In the web-based TNC use case, the task of issuing tokens that contain the TNC check result is performed by the VSP. In the following description, the VSP therefore takes on the role of the WS-Trust STS. According to the WS-Trust specification [Nad07], the first RST message can already contain negotiation or challenge information. This initial RST message can therefore contain the initial TNCCS-Batch generated by the TNC Client.

   All WS-Trust messages are regular SOAP messages. They can therefore be secured using the mechanisms described in WS-Security. In the attestation-based TNC model, the client can obtain a security token from the SP to authenticate itself to the VSP. If such a token has been received, it can be attached to the RST using the mechanisms described in WS-Security to authenticate the request. Alternatively, if the user performs the authentication directly with the VSP, the client can generate a token (for example, a username token) and attach it to this message.

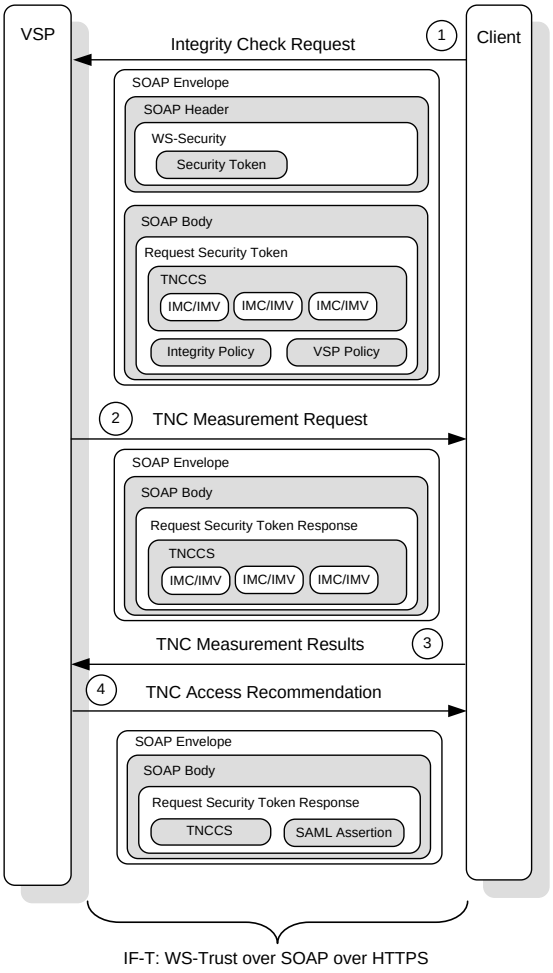


Figure 5.10: Realising IF-T with WS-Trust

2. The response returned from the VSP (RSTR message) contains a TNCCS-Batch created by the TNC Server that contains measurement instructions for IMCs on the client.

3. If the message exchange is not yet complete, that is, if more TNCCS Batches need to be exchanged, the requestor (i.e. client) can send an RSTR message which contains additional information. Typically, a TNC Client would send integrity measurement results to the TNC Server in this step of the protocol.

4. This process repeats at step 2 until the TNC message exchange is completed, that is, an access recommendation can be made or an error occurs resulting in a SOAP fault message. If the TNC Server was able to produce an access recommendation, a security token (in form of a SAML attribute assertion) is sent within the final *Request Security Token Response (RSTR)* message. The requestor is now in possession of the token and can use it to access a service.

As described above, a TNC message exchange can be realised using a WS-Trust message pattern. Two extension mechanisms of WS-Trust were leveraged for TNC. Firstly, the extension mechanism which allows the inclusion of additional information in RST and RSTR messages has been used to transfer TNCCS Batches and policies. Secondly, the negotiation and challenge extension is used that allows multiple round-trips to be performed before a token is issued, thus enabling the TNC integrity exchange. Client authentication can be performed using standard WS-Security methods.

Using the approach described above, TNCCS Batches are tunnelled through a WS-Trust message exchange. WS-Trust is used to issue a security token based on the TNC measurements and the SP's policies. All artefacts, that is, SAML tokens, policies, and TNCCS messages, are encapsulated using standardised methods.

5.4 Summary and Conclusion

In this chapter two different approaches to implementing the TNC-based integrity check have been discussed. A TLS-based approach, as described in section 5.2, has the advantage of being broadly applicable not only in web-based applications but also in other use cases such as email. Sections 5.2.2 and 5.2.3 analyse whether existing extension mechanisms for the TLS protocol can be used as the transport mechanism for a web-based TNC check. Both mechanisms provide the possibility of including EAP messages in the TLS handshake protocol. However, both mechanisms fall short in providing a possibility for including additional artefacts, such as a SAML token. Additionally, implementation issues exist that can complicate a TLS-based TNC check.

The second approach discussed in this section, which overcomes the shortcomings of a TLS-based approach, is based on application layer protocols. Sections 5.3.1 and 5.3.2 briefly describe two base protocols, that is HTTP and SOAP, respectively. HTTP is the base protocol for almost all web-based communication. SOAP, building upon HTTP, provides a standardised container format for exchanging XML documents. Both protocols, however, do not provide a mechanism for semantically wrapping information that is exchanged. This semantic wrapping can be provided by a message format that is used on top of SOAP. In this chapter, two existing message formats have been considered for encapsulating the TNC message exchange. The specifications of the SAML protocol (cf. section 5.3.3) are too restrictive for the TNC check use case. It is not possible to extend the simple request/response-based message exchange in order to enable multiple round-trips as required by TNC.

WS-Trust (cf. section 5.3.4) does not have this limitation. Section 5.3.4 describes how WS-Trust can be adapted for the requirements of a web-based TNC check by exploiting two of its extension mechanisms. This approach allows the performance of the TNC check within a standardised WS-Trust message exchange. The general intention of WS-Trust is to pro-

vide a mechanism that allows the request for and receipt of security tokens, such as SAML assertions. Typically, a token issuing party requires the requesting party to provide an authentication token before issuing a token. The idea presented in this chapter is to extend this scenario for the TNC use case by basing the token issue on the integrity state of a client and the integrity policy of an SP.

The resulting protocol stack is depicted in figure 5.11. At the outermost layer, HTTPS provides a TLS/SSL secured communication channel. Because HTTP(S) is commonly used in web-based environments, it is unlikely that it will be blocked by firewalls or web proxy server. In the next layer, SOAP provides an encapsulation mechanism for XML data. In addition, this layer provides the possibility to authenticate the sender of a SOAP message by using methods defined in WS-Security. WS-Trust, which uses SOAP, has two responsibilities. Firstly, it provides a mechanism for requesting and returning security tokens. As described in chapter 4, the TNC result is encapsulated in a SAML attribute statement that is part of a SAML security token. Secondly, WS-Trust provides an encapsulation mechanism for TNCCS-Batch messages and policies.

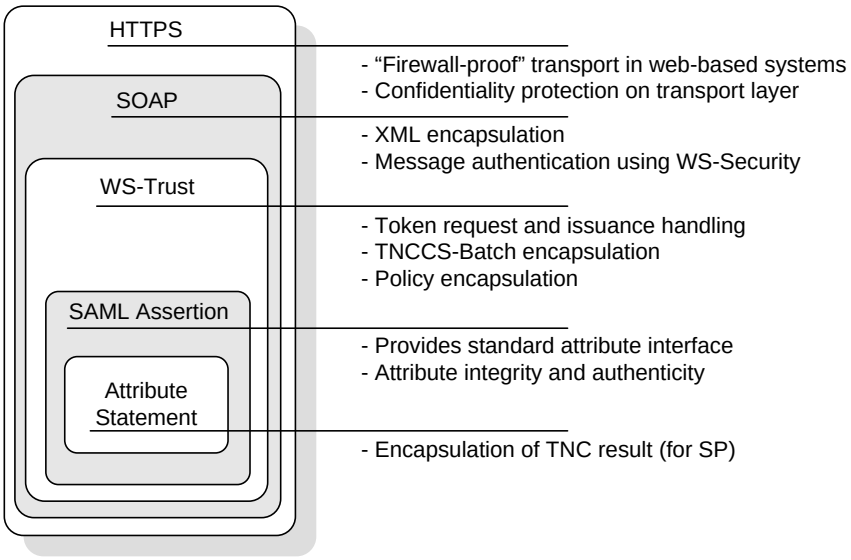


Figure 5.11: Proposed protocol stack for realising IF-T in web-based environments

A TNC integrity check is typically triggered by an SP. Before a client can use the WS-Trust-based mechanism to perform the TNC check, it needs to obtain artefacts from the SP (that is, VSP policy, integrity policy, and (optionally) an authentication token). The next chapter discusses how these artefacts can be obtained and how the TNC check is triggered, that is, how the IF-PAA interface can be realised.

Chapter 6

Requesting Integrity Checks over IF-PAA

In the previous chapter, mechanisms for performing a TNC check between a client and a trusted VSP were evaluated. WS-Trust was identified as a message format capable of encapsulating TNC messages in a standardised way. Before the TNC check can be performed, the attestation-based architecture requires that a TNC check is triggered by an SP. When the TNC check is finished, the client is expected to return the TNC check result to the SP. The interface between client and SP is called IF-PAA (*Interface - Policies/Authentication/Assertion*) as it is used to transfer *policies* (integrity and VSP policy), an *authentication* statement (e.g. a SAML assertion issued by the SP), and an *assertion* issued by the VSP for the SP containing the TNC integrity check result. This chapter proposes a mechanism that fulfils the requirements of IF-PAA, which are elaborated on in the following.

6.1 Requirements

IF-PAA is not defined within the TNC specification. It needs to be introduced for the web-based TNC use case because the flow of information in the assertion-based model is different from the original TNC architecture[1]. The interface has two purposes.

Returning TNC check result Firstly, it is responsible for delivering an assertion about the integrity state of the client to an SP. While IF-PAA is not defined within the TNC specifications, the information that needs to be included in this assertion is described in [Tru07g, section 2.8 seqq.]. A mapping between this information and a SAML assertion was proposed in section 4.3.3.

Initiation of TNC check Secondly, IF-PAA defines how an SP can trigger a TNC check. When triggering the TNC integrity check, the SP can provide the following information for the client, as described in section 4.4: the integrity policy of the SP, a security token for authenticating the user at the VSP, and the VSP policy describing which VSPs are trusted by the SP. Since a user relies on a standard Web browser for accessing the SP, the SP must trigger the TNC check through the browser.

[1] Cf. section 3.3.3

Limitation in choice of protocols In the previous chapter, SOAP-based message formats such as WS-Trust have been introduced for encapsulating and transporting similar artefacts. However, these mechanisms cannot be used for IF-PAA, because a standard Web browser is used as the client software for IF-PAA. By default, standard Web browsers are not capable of using Web service based protocols, such as SOAP and WS-Trust. The design of IF-PAA must therefore take the limited capabilities of a Web browser into account.

Although IF-PAA cannot be based on the same mechanisms as IF-T, both interfaces have a common requirement[2]. Only HTTP and TLS-based protocols can be considered for both IF-T and IF-PAA to prevent TNC related messages from being blocked by firewalls or proxy servers. In IF-PAA this requirement is even more important, as clients use standard browsers to communicate with web-based applications (that is, with SPs). This communication is almost exclusively based on HTTP or HTTPS. As the TNC check needs to be integrated into existing systems, it has to be based on one of the aforementioned protocols. After the TNC check is triggered, client software (for example a browser plug-in) can be invoked which performs the TNC check using WS-Trust, as described in the previous chapter.

Open standards As discussed previously, making use of existing standards and their extension mechanisms is preferred to developing new mechanisms in order to ease the integration of a web-based TNC integrity check into existing systems. Finally, the mechanism chosen for IF-PAA shall not restrict an SP's choice of authentication methods for its clients.

In the following, two approaches are discussed. Section 6.2 introduces a TLS-based approach, followed by a discussion of an application layer approach in section 6.3.

6.2 Analysis of an SSL/TLS Approach

The TLS extensions that have been discussed for IF-T in the previous chapter deal with *authentication* for legacy systems. In contrast, IF-PAA's main responsibility is to transfer information that is used in the *authorisation* process. Authorisation, however, does not lie within the scope of the protocol extensions that have been discussed in section 5.2 (such as TLS/EAP). The only relevant TLS mechanism that deals with authorisation is the *TLS Authorization Extension* [MB06], which is described in the following.

[2]Cf. section 5.1 for the requirements of IF-T.

6.2.1 TLS Authz - TLS Authorization Extensions

The *TLS Authorization Extension (TLS Authz)*[3] is a TLS extension for exchanging authorisation data. An overview of this extension is given in the following and depicted in figure 6.1.

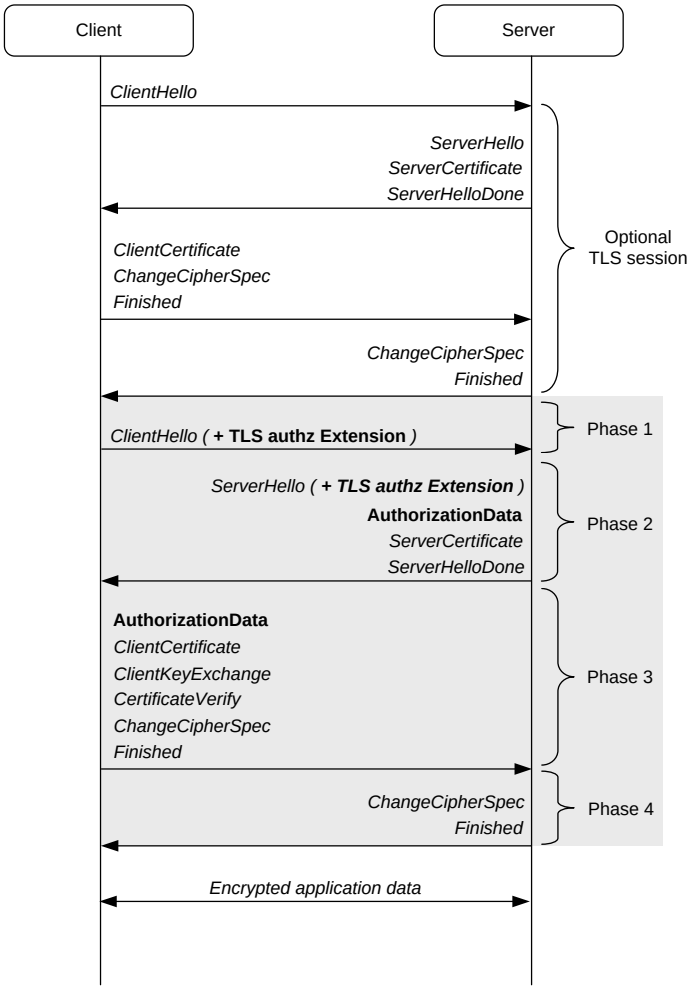


Figure 6.1: TLS handshake using the TLS Authorization Extension

TLS provides a mechanism for performing mutual authentication by using client and server X.509 certificates. Similarly, TLS Authz provides the possibility of performing a mutual authorisation. It allows the client to send authorisation data to the server as well as allowing a server to send authorisation data to a client. The data sent to the server can contain information that allows the server to determine the access level granted to a client. According to [MB06, section 3.2], authorisation data sent from the server to the client is expected to contain information such as the server's qualifications, reputation, or accreditation. This allows a client to decide whether and how it will communicate with this server.

[3]This Internet draft is specified in [MB06].

Two data formats for expressing authorisation information have been incorporated into the TLS Authz extension: SAML assertions and X.509 attribute certificates[4]. An authorisation statement that uses one of the aforementioned formats is included into the TLS handshake to provide additional information for the authorisation decision process. This approach is depicted in figure 6.1. This figure shows two alternative protocol flows. The core TLS Authz message exchange is depicted with a grey background colour and operates as follows.

In phase 1, the client indicates its support for the TLS Authz extension by including the corresponding extension type[5] in the *ClientHello* message. In this message, the client also indicates which of the two formats (SAML assertion and/or X.509 attribute certificates) it supports.

In phase 2 the server can express its support for the TLS Authz extension in the *ServerHello* message. Using the TLS handshake extension mechanism, the server can send supplementary data within the TLS handshake. The TLS Authz extension uses this mechanism to send an authorisation statement to the client (which, for example, contains a statement about the server's qualifications). This mechanism can be adapted for the TNC use case. Instead of sending information about the server, this statement could contain the authentication confirmation for the VSP, which asserts that the client has successfully performed an authentication with the SP.

The client can provide its authorisation statement in phase 3 using the same supplemental data mechanism as above. This step of the protocol can also be leveraged for a web-based TNC check, that is, the statement could contain the TNC check result.

TLS Authz, as described so far, provides a means of exchanging authorisation statements in the form of SAML assertions or X.509 attribute certificates. These authorisation statements are not encrypted and are therefore subject to eavesdropping attacks. To protect the confidentiality of the message exchange, a transport security mechanism can be used. TLS itself provides such a protection. However, as shown in figure 6.1, the authorisation statements are exchanged before the *ChangeCipherSpec* message is sent, that is, before encryption is enabled. To enable an encrypted message exchange at the transport layer, an additional TLS handshake needs to be performed before the TLS Authz enabled TLS handshake. This additional handshake is labelled *Optional TLS session* in figure 6.1. Thus, the communication between server and client is already encrypted when the *ClientHello* message in phase 1 is sent. During this upstream handshake, a mutual authentication (for example using X.509 certificates) can be performed before authorisation data is exchanged.

[4] As described in section 4.3.1

[5] Cf. section 5.2.1, in which an overview of the TLS handshake extension mechanism is given.

6.2.2 Summary and Verification of Suitability

The TLS Authz extension is capable of transporting SAML assertions using the TLS handshake protocol. It could thus be used for exchanging SAML assertions between an SP and a client. A TLS Authz based mechanism for triggering a TNC check and returning the result of this check to the SP can consist of the following steps.

Mutual Authentication A TLS-based mechanism, for example based on X.509 certificates, can be used to authenticate the parties.

Obtaining Authentication Proof While not specifically intended by TLS Authz, this proof can be obtained by the client in phase 3 of the protocol. In this step an SP can produce a SAML assertion confirming a successful authentication of the client.

TNC Integrity Check The TNC check is performed outside the scope of the TLS Authz protocol described in section 5.3.4.

Providing TNC Check Results The client can forward the SAML assertion obtained from the VSP to the SP using the mechanisms specified in TLS Authz. The SP can then consider the contents of the SAML assertion during its authorisation decision process.

Exchanging Application Data After all authorisation statements have been exchanged, the TLS handshake is completed and application data can be exchanged.

While the above list fulfils some of the requirements stated in section 6.1, it does not meet all of them.

Firstly, TLS Authz does not define procedures for specifying which authorisation data is expected to be included in an authorisation statement. Instead, the TLS Authz specification states that the principle of least privileges shall be applied to determine the set of authorisation data to be sent [MB06, section 3.1], that is, sending as little information as necessary. This approach cannot be aligned with the requirements of IF-PAA. A VSP needs to check the integrity state against an integrity policy as it is otherwise unable to derive an access recommendation. If the integrity policy cannot be exchanged as part of the TNC check request, it could be exchanged before a TNC check using an out-of-band mechanism. Such an approach might be feasible in a closed environment, in which the policies associated with a specific SP are predifined. However, this approach fails in an open environment, in which the VSP and SP are independent of each other. This scenario would also require the VSP to know the identity of the requesting SP[6], which can result in privacy issues as discussed in section 4.5.

Secondly, because of limitations of TLS Authz, an SP cannot indicate which VSPs are considered trustworthy for performing the TNC integrity check. That is, it cannot include the

[6]This only applies if a VSP serves more than one SP. However, a scenario where a VSP is limited to creating access recommendations for only one SP also contradicts the idea of a TNC check in an open environment.

VSP policy in the TNC check request. Without this policy, the VSP/SP relationship needs to be known by the client a priori and cannot be handled in a flexible manner during a TNC integrity check. While such an approach might be feasible in a closed and managed environment, it is not suitable in an open web-based environment.

Thirdly, TLS Authz raises implementation issues similar to those that have been discussed in section 5.2.4, where a TLS-based mechanism was considered for IF-T. The underlying TLS infrastructure at the SP needs to be adopted when TLS Authz is used to enable TNC integrity checks. The TLS tunnel is terminated at an SSL module or SSL accelerating hardware. The result of a TLS-based integrity check, however, has to be accessible at the application layer to enforce the (application specific) TNC access decision. It would thus be necessary to create a TNC specific interface between the SSL software or hardware and the policy enforcing entity.

Fourthly, TLS Authz limits the choice of authentication mechanisms that can be used by the SP. This creates a conflict with the requirements for IF-PAA, which states that the TNC check must not impose restrictions on the authentication mechanism that can be used in conjunction with TNC. Typically, authentication is performed before authorisation information is exchanged [SG07][7]. As the TNC check result is treated as authorisation data by SPs, it follows that clients need to be authenticated before a TNC check can be performed. Authentication in a TLS Authz-based approach can only be realised using authentication methods that rely on TLS, such as an X.509 certificate based mechanism. This prevents authentication methods on the application layer from being usable as the first factor in an authentication process. Consequently, popular approaches, such as HTML form based username/password logins, would not be usable when relying on TLS Authz for triggering the TNC check.

As mentioned before, authentication is performed before authorisation. By moving the TNC check result to a higher protocol layer, and thereby to a later stage in the message exchange, more methods for performing the authentication become available. If the TNC check result is performed using an application layer protocol, all methods of authentication that are implemented in an equal or lower protocol layer can be used. That is, by moving IF-PAA to the application layer, authentication methods on the application layer (for example username/ password based HTML forms or token-based authentication in federated identity systems) can be used as well as methods in lower protocol layers, such as TLS-based X.509 certificate or layer 3 authentication. An approach in which IF-PAA is realised using application layer mechanisms is discussed in the following section.

[7] Exceptions are systems that rely on anonymous authorisation. In these systems, authorisation information is self contained and does not require an identifier for the user [AVKK+04].

6.3 Analysis of an Application-layer Approach

As discussed in the previous section, using a TLS-based approach limits the choice of authentication mechanisms that can be used between client and SP. One way of overcoming this limitation is to make the TNC check result available at the highest protocol layer, that is, at the application layer. This section proposes an approach for implementing IF-PAA at this layer.

As stated in section 6.1, the mechanisms available for realising IF-PAA are limited to those that are supported by a standard Web browser. The main purpose of Web browsers is to send HTTP requests to a remote server in order to fetch, interpret, and display HTML documents[8]. As discussed in section 5.3.1, HTTP cannot be used to encapsulate information in a semantically meaningful way. In the following, section 6.3.1 discusses how HTML can be leveraged for the purpose of triggering the TNC check. Section 6.3.2 elaborates on how an SP can issue an authentication token for the client. Policy formats for expressing VSP and integrity requirements are proposed in section 6.3.3. Finally, section 6.3.4 combines the results of the previous sections and proposes a mechanism for realising IF-PAA.

6.3.1 HTML-encoded TNC Check Request

In the web-based TNC scenario, the TNC integrity check is triggered through a Web browser. In addition, the browser delivers the TNC check result from a client to an SP. A Web browser is unaware of the actual TNC check as it is performed in an external process or program that is started after the browser has detected a TNC check request. Starting a TNC check is thus similar to starting an external application that is used to display external or interactive content, such as multimedia files or Java applets. A feasible approach is thus to embed the TNC check request in an HTML document and to treat it as an invocation of an external application for handling the TNC check. This approach is discussed in the following.

6.3.1.1 Object Tags

In addition to plain text, an HTML document can include other types of content, such as images, movies, or embedded programs. HTML 4 introduces the *Object* element [RHJ99, section 13], which offers a generic way of including external objects in an HTML page. If a browser has support for the content type that is embedded via an *Object* element, it launches an external application to display the external objects (for example a PDF document or video file). Additional information can be passed to such an external application using parameter tags. Listing 6.1 shows a simplified example, in which a Windows Media Video

[8]Other tasks include, for example, executing client side scripts, such as Javascript.

(wmv) file is embedded in an HTML page. When the page is being rendered by the browser, it starts the video player associated with the object type *video/x-ms-wmv* and passes the parameters encoded within *param*-Tags to that application (that is, the path of the movie file and the control instructions in the example).

Object tags can further be used to embed *active* content, such as Java applets or ActiveX controls into a HTML page. Java applets and ActiveX controls are programs that are downloaded and executed on a client's machine. In the past, a number of vulnerabilities in the interpretation of the object tags in some browser software[9] were discovered. Furthermore, executing potentially malicious applications that are downloaded from the Internet is associated with certain security risks. Consequently, object tags cannot reliably be used as they are disabled by high security browser settings [Nan07]. An alternative to object tags that is not affected by browser security settings is described next.

```
<html>
 ...
 <object type="video/x-ms-wmv" width="340" height="280">
  <param name="src" value="http://www.exmaple.com/path/to/movie.wmv" />
  <param name="autoStart" value="true" />
  <param name="canSeek" value="true" />
  <param name="mute" value="0">
 </object>
 ...
</html>
```

Listing 6.1: Using an object tag to embed a video file into a HTML page

6.3.1.2 XHTML-Tags

XHTML[10] is a W3C recommendation that combines HTML and XML. More precisely, XHTML combines the mark-up language features of HTML with XML's syntax extensibility. XHTML can thus be used to embed additional information within a Web page using XML syntax features.

The XHTML extension capabilities have been used in the past for a number of different purposes. Examples include the *Mathematical Markup Language (MathML)* [CIMP03] which is used to visualise mathematical formulae in browsers. Similarly, the XML-based *Scalable Vector Graphics (SVG)* format allows embedding of vector graphics into HTML documents [BB04]. More recently, XForms [BDK+07] was specified to extend HTML form functionality. In order to ensure that a web page that includes extended XHTML commands can still be rendered by standard browsers, conformance rules must be obeyed [APM+06, section 3].

[9] Cf. for example *OBJECT Tag Vulnerability* (http://www.cert.org/advisories/CA-2000-16.html) and *OBJECT tag memory corruption* (http://xforce.iss.net/xforce/xfdb/25978).

[10] Originally specified in http://www.w3.org/TR/2001/REC-xhtml11-20010531 and updated in [AM07b].

These conformance definitions require that the syntax definition[11] is referenced in the XHTML page.

Unlike with *Object*-tags, a browser cannot be configured to start an application when they encounter a certain XHTML fragment. Instead, it must either have inbuilt support for a certain XHTML format[12], or a browser plug-in handles the processing of such an XHTML fragment in lieu of the browser.

Such an approach, in which a browser plug-in picks up an XHTML fragments and starts an external application, is used in the identity management system Cardspace[13]. The XHTML-based approach is further described in the following section using Cardspace as an example of an implementation thereof.

6.3.1.3 Launching Applications from XHTML: An Example Application

Cardspace is an architecture for digital identity management and authentication. A web-based TNC integrity check and the Cardspace identity management system are not conceptually related; however, both systems exchange similar artefacts in a similar environment.

The main idea of Cardspace is very similar to the idea of WS-Trust, which has been described in section 5.3.4. In fact, parts of Cardspace's communication is based on WS-Trust (among other WS-* standards). In WS-Trust as well as in the Cardspace model, a service providing party (called a *Relying Party (RP)* in Cardspace) does not perform an authentication itself. Instead, it relies on a third party to perform this authentication. This third party, called an *Identity Provider (IdP)*, issues a security token to the user which is evidence of a successful authentication with the IdP. In Cardspace, this security token further contains *claims* that state user identifiers (e.g. username) or attributes of the user (e.g. his or her age). Figure 6.2 summarises this communication model [BSB08, page 181 seqq.].

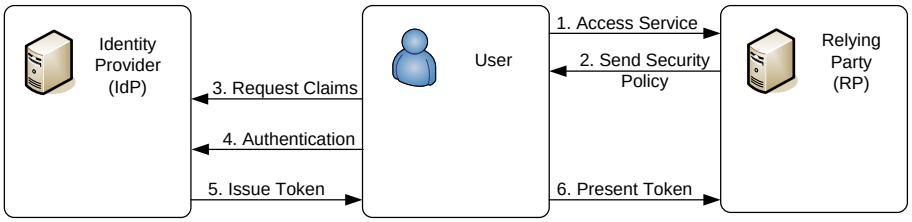


Figure 6.2: Simplified communication flow in the Cardspace model

A difference between WS-Trust and Cardspace is that Cardspace can be used in web-based environments, that is, where a browser is used to access a service. Cardspace requires that

[11] Expressed using the *Document Type Definition (DTD)* format (http://www.w3.org/TR/REC-xml/#dt-doctype). Further work to enhance the process of validating documents that include extended XHTML has been undertaken in [KN06].

[12] As it is often the case with the Scalable Vector Graphics format.

[13] See http://msdn.microsoft.com/CardSpace.

the client is equipped with additional software. This software, called the *Identity Selector*, is responsible for interacting with the user and performing the WS-* based communication with an IdP. It is therefore necessary to bridge the communication between a browser (operating on a HTTP and HTML level) and the client software (operating on a Web service level). In Cardspace, this bridging is achieved by encapsulating information using XHTML[14]. The XHTML representation is parsed by a browser (or a browser plug-in) and passed on to the client software. This mechanism is called the *Identity Selector Interoperability Profile (ISIP)* [Nan07]. Listing 6.2 shows an example of an XHTML fragment as defined by ISIP.

```
1  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:ic>
2   ...
3   <form method="post" action="processIC.aspx">
4    <ic:informationCard name="xmlToken"
5     issuer="http://exampleIssuer"
6     tokenType="urn:oasis:names:tc:SAML:1.0:assertion">
7      <ic:add claimType="http://schemas.xmlsoap.org/.../emailaddress"
8       optional="false" />
9    </ic:informationCard>
10  </form>
11   ...
12 </html>
```

Listing 6.2: Using the XHTML syntax described in the Identity Selector Interoperability Profile

The XHTML tags used by Cardspace have a namespace prefix (*ic*) to distinguish them from standard XHTML tags. The information that needs to be passed from the RP to the user are encoded in lines 4–9 using standard XML syntax. Attributes of the root element (*ic:informationCard* in line 4) describe which IdP shall issue the requested token (line 5) and which token type is expected (line 6). Furthermore, an RP can state which claims shall be contained in the requested token. In the example, line 7 specifies that the subject's email address is required. The Cardspace XHTML tags in the example are enclosed in an HTML form. This form is used to send the token obtained from an IdP to the RP via an HTTP POST message[15].

The approach used by Cardspace provides a way of invoking the client software using a browser and standard XHTML. In addition, this approach allows an RP to pass information to this client software, which in turn sends back the requested token. However, the syntax used in Cardspace cannot be used to realise IF-PAA, as it targets a different scenario. For example, the claim focussed policy approach cannot be applied to integrity policies. However, the general mechanism of encoding information in XHTML which is parsed by a browser extension and forwarded to an external application is applicable to the TNC use case.

[14]Cardspace also supports an *object tag* based approach. However, this approach is not further discussed due to its drawback outlined in section 6.3.1.1

[15]Cf. section 5.3.1.

In the following sections, an XHTML syntax will be developed that takes the requirements of IF-PAA into account.

As described earlier, an SP can produce an optional authentication token that a client can present at the VSP for authentication purposes. WS-Trust, as introduced in section 5.3.4, provides the means for issuing tokens. IF-PAA, however, cannot make use of Web service-based protocols such as WS-Trust. The following section describes an approach that allows WS-Trust to be used in a browser environment.

6.3.2 Using WS-Trust in a Browser Environment

A client that has been issued with a security token from an SP can use this token to perform an authentication with the VSP. As outlined previously, various security token formats exist. An example of such a format is a SAML authentication assertion, which was briefly described in section 4.3.3. The token format affects how a client uses the token for performing an authentication with a VSP. It is therefore necessary to indicate the token format to the client. WS-Trust, as introduced in section 5.3.4, provides a standardised SOAP-based protocol for encapsulating a security token and its metadata. However, because browser applications do not support SOAP, it cannot be used directly for IF-PAA. In order to enable WS-Trust in browser environments, RST and RSTR messages need to be embedded into (X)HTML. The *Web Services Federation Language (WS-Federation)* [LAB+06] standardises this method.

WS-Federation describes how federated identity scenarios, in which identities are shared amongst different organisations, can be realised. In contrast to SAML, which also provides support for federated identities[16], WS-Federation is built on top of WS-Trust. Amongst other scenarios, WS-Federation describes how federation can be realised (using WS-Trust) in a browser-based environment. This mechanism is specified in [LAB+06, section 13] as the *Passive Requestor Profile*.

At its core, the Passive Requestor Profile uses the same message exchange pattern and message formats as WS-Trust[17]. However, instead of relying on SOAP, it uses HTTP(S) and HTML forms for requesting and issuing security tokens. A token is requested by using a specially crafted HTTP request, which carries an RST message as a parameter. The answer to such a request, that is, an issued token, is encapsulated using the RSTR message format and is placed into an HTML form. By submitting the form, the token is transferred to the service offering party inside the RSTR message.

This mechanism provides a way to issue a security token to a browser using a message format defined in WS-Trust. The approach, as outlined above, is applicable to a web-based

[16] Cf. section 4.3.2.
[17] Cf. section 5.3.4.

TNC check. An SP that issues a token for a client can encapsulate the token and its metadata in an RSTR message. This message is placed into an HTML document that is sent to the client's browser. The client can extract the token and use it for performing an authentication with the VSP.

While it is useful to encapsulate the security token used to perform an authentication with the VSP, it is not necessary to encapsulate the security token that contains the TNC check result. The authentication token needs to be incorporated by the client into the SOAP-based message exchange. However, the token containing the TNC check result is simply passed on to the SP and therefore does not to be encapsulated.

The outcome of a TNC check depends on the client's integrity state and the integrity requirements that an SP has. The following section proposes policy formats that can express these integrity requirements and also the trust relation between SPs and VSPs.

6.3.3 Expressing Policies

As outlined in section 4.4, policies play a vital role in the assertion-based architecture. They allow an SP to state which VSPs are trusted (VSP policy) and which integrity requirements a client must fulfil in order to be granted access to a service (integrity policy). This section proposes formats for expressing these policies.

The policy mechanism in Cardspace, which was introduced in the previous section, cannot be applied to a web-based TNC check. It is too limited and cannot be customised for the web-based TNC check use case. It is therefore necessary to create a policy mechanism that is specific to TNC. Instead of inventing a new policy language, an existing language shall be leveraged for the web-based TNC use case.

In the following section existing policy standards are discussed. An appropriate standard is selected and used to develop two policy formats. Section 6.3.3.2 proposes a format for expressing an integrity policy. A VSP policy format is proposed in section 6.3.3.3.

6.3.3.1 Policy Languages

Policies are commonly used for expressing requirements and capabilities of services. Two policy definition languages can be considered as de facto standards for expressing policies in Web applications [KMW08]: The *extensible Access Control Markup Language (XACML)* [Tim05] and the *Web Service Policy Framework (WS-Policy)* [VOH+07] .

XACML focusses on access control mechanisms for web-based resources. It describes resources, actions, rules, and permissions for making authorisation decisions [Tim05]. As such, an SP might use XACML internally in its access decision process in which the TNC

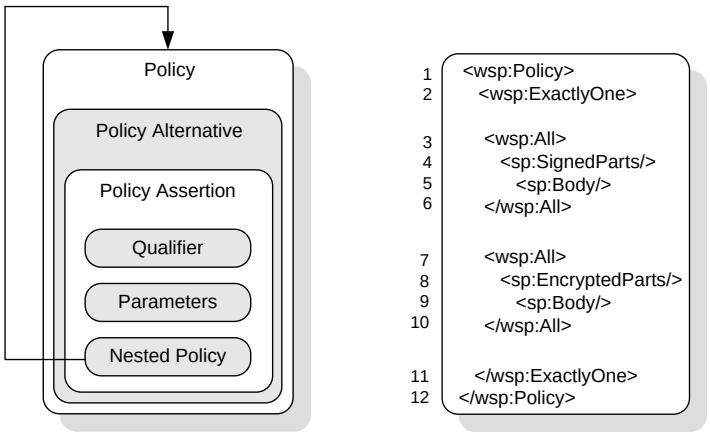


Figure 6.3: Schematic overview of WS-Policy (a) and example of a policy written in WS-Policy (b)

check result is used as one input paramater. However, because of its focus on access control mechanisms, XACML is not suitable for expressing integrity and VSP policies.

Compared to XACML, WS-Policy provides a more general approach for expressing policies [KMW08]. Policies described using WS-Policy are used to communicate the requirements and capabilities of a Web service. Instead of defining a specific language, WS-Policy defines an XML-based framework for the assembly of domain-specific policy languages. Existing domain specific languages include WS-SecurityPolicy [NGG+07] which deals with secure communication parameters and WS-PolicyAssertions [BHK+03] which covers text encoding and language support for Web services. The policies used in the web-based TNC use case can be considered to be such domain specific languages, as they are specific to the TNC use case.

The schema of WS-Policy is depicted in figure 6.3a[18]. It consists of three entities. The outer element, that is the *policy*, contains one or more *policy alternatives*. Each policy alternative contains a *policy assertion*. This policy assertion represents a requirement or a capability that is identified by a qualifying name and optional parameters. Furthermore, a policy assertion can contain a nested policy, which defines further details about the policy assertion.

WS-Policy defines three operators for nesting policies, policy alternatives, and policy assertions in the *policy normal form*. The element <*wsp:Policy*> initiates the beginning of a new policy block. A policy alternative is enclosed by <*wsp:All*>. To group policy assertions into a policy alternative, the element <*wsp:ExacltyOne*> is used. Figure 6.3b shows an example policy that makes use of these elements and combines them with elements from WS-SecurityPolicy. The example policy defines two policy alternatives (lines 3–6 and lines 7–10). These alternatives are combined using the <*wsp:ExactlyOne*> operator (lines 2 and 11).

[18] Based on [VOH+07].

An interpretation of this policy statement would be that for invoking a Web service it is necessary to either sign or encrypt the SOAP message body.

By using the mechanism described above, it is possible to define policies in a flexible manner. WS-Policy allows for the definition of custom tags to express domain specific policies. This approach is used to create a policy language for expressing the integrity requirements, as described in the next section.

6.3.3.2 Integrity Policy

As described in section 4.4, an SP can express its integrity requirements using a high level integrity policy. This policy is sent to the VSP, where it is translated into an IMV-specific policy language. The IMVs installed on the VSP compare the integrity state of the client with the obtained integrity policy. The result of this comparison is sent back to the SP where it is used during the SP's access decision process.

The high level integrity policy contains general requirements for a client's integrity state. Its level of abstraction is similar to the abstraction level found in the terms of use of an online banking portal or an employee's contract, as described in chapter 1. As such, they are very general and do not require the SP to have specialist knowledge or keep up-to-date on updates for security products, such as virus scanners.

Ideally, an integrity policy language covers all aspects of security and other requirements that an SP can have for its clients. However, the definition of such an exhaustive language is not within the scope of this thesis. Instead, a base format is proposed that can be used to create an integrity policy language. Based on the proposed format, a set of policy instructions have been defined to express commonly found integrity requirements. While the proposed policy format does not cover all possible integrity requirements, it can be used to create sufficiently complex policies that can be used in a test bed approach for studying the behaviour of a web-based TNC check[19]. The base format and the set of requirements, which are shown in listing 6.3, are described in the following[20].

As with other domain specific languages, such as WS-SecurityPolicy, the language for expressing the integrity policies has its own XML namespace declaration. All elements that are specific to the integrity policy are therefore prefixed with *wtip (web-based TNC integrity policy)*. By applying the WS-Policy syntax, integrity policy alternatives can be nested. Rather than explicitly stating a policy, it is possible to refer to a policy using an ID, a name, or a URL. In such an approach it is necessary that the policy that is referred to is accessible to both the VSP and the SP. The first policy alternative *<wtncp:WindowsOperatingSystem>* in lines 5–16 states that all nested policies only apply for clients with a Microsoft Windows operating system. These clients must have an activated personal firewall (lines 8–10) and

[19] Cf. chapter 7.
[20] The XML Schema of the proposed format can be found in section D.2.

```
1  <wtip:IntegrityPolicy id="..." name="..." URI="...">
2    <wsp:Policy>
3      <wsp:ExactlyOne>
4        <wsp:All>
5          <wtip:WindowsOperatingSystem>
6            <wsp:ExactlyOne>
7              <wsp:All>
8                <wtip:PersonalFirewall>
9                  <wtip:active/>
10               </wtip:PersonalFirewall>
11               <wtip:Antivirus>
12                 <wtip:uptodate/>
13               </wtip:Antivirus>
14             </wsp:All>
15           </wsp:ExactlyOne>
16         </wtip:WindowsOperatingSystem>
17       </wsp:All>
18     </wsp:ExactlyOne>
19   </wsp:Policy>
20 </wtip:IntegrityPolicy>
```

Listing 6.3: Example policy for demonstrating the integrity policy structure

up-to-date anti-virus software (lines 11-13) installed. Using the WS-Policy semantics, variations of this policy can easily be created. By defining new elements in the <*wtip*> namespace, the vocabulary of the integrity policy can be extended and new integrity requirements can be expressed.

In addition to the integrity policy, an SP must also include a VSP policy in the TNC check request. The format of this policy is described next.

6.3.3.3 VSP Policy

An SP uses the VSP policy to state which VSPs it trusts to perform the TNC check. A client can use this information to choose a VSP that is trusted by an SP, that is, an VSP from which an SP accepts the TNC check result. As described in section 4.4, trust relations can be expressed in two ways:

Explictly by stating the URLs of trusted VSPs.

Implicitly by stating a Certificate Authority (CA) that is trusted to issue certificates only to those VSPs that perform the TNC check to a predefined standard or offer predefined capabilities.

Both mechanisms define (explicitly or implicitly) which VSPs are trusted for issuing a security token that contains the TNC check result. Mechanisms for defining trusted issuers are also part of WS-SecurityPolicy. Instead of redefining these mechanisms for the TNC use case, a subset of the WS-SecurityPolicy syntax is reused for the VSP policy. Listing 6.4

```
1  <wtvp:VSPPolicy>
2   <wsp:Policy>
3    <wsp:ExactlyOne>
4     <wsp:All>
5      <sp:IssuedToken sp:IncludeToken="http://.../IncludeToken/AlwaysToRecipient">
6       <wsp:ExactlyOne>
7        <wsp:ExactlyOne>
8         <sp:Issuer>
9          <wsa:Address>https://trustedVSP.com/TNCService</wsa:Address>
10         <wtvp:VSPName>Secure Issuer</wtvp:VSPName>
11        </sp:Issuer>
12        <sp:Issuer>
13          <wsa:Address>https://vsp.company.com/TNCService</wsa:Address>
14         <wtvp:VSPName>Local VSP</wtvp:VSPName>
15        </sp:Issuer>
16       </wsp:ExactlyOne>
17       <sp:RecipientSignatureToken>
18        <sp:X509Token sp:IncludeToken="http://.../IncludeToken/AlwaysToRecipient">
19         <wsp:ExactlyOne>
20          <sp:IssuerName>CN=wTNC Certification Authority A</sp:IssuerName>
21          <sp:IssuerName>CN=wTNC Certification Authority B</sp:IssuerName>
22         </wsp:ExactlyOne>
23        </sp:X509Token>
24       </sp:RecipientSignatureToken>
25      </wsp:ExactlyOne>
26     </sp:IssuedToken>
27    </wsp:All>
28   </wsp:ExactlyOne>
29  </wsp:Policy>
30 </wtvp:VSPPolicy>
```

Listing 6.4: Example policy for demonstrating the VSP policy structure

shows an example VSP Policy to demonstrate the resulting syntax[21]. All elements that have been reused from WS-SecurityPolicy are prefixed with *sp*. WS-SecurityPolicy makes use of *Web Services Addressing (WS-Addressing)* [GHR06]. WS-Addressing provides semantics for expressing address information of Web service endpoints. All elements from the WS-Addressing namespace are prefixed with *wsa*. Finally, VSP policy specific elements use the prefix *wtvp (web-based TNC VSP policy)*.

The example policy above includes both explicit and implicit requirements. These requirements are nested in a <*sp:IssuedToken*> element that has a parameter (*sp:IncludeToken*) (line 5). Using WS-SecurityPolicy syntax, this parameter states that the security token that contains the TNC check must be sent to the SP; that is, other options, such as referencing the token via a URL, are not supported.

Lines 7–16 contain explicit trust relation instructions. Two trusted VSPs are listed (lines 8-11 and lines 12-15). The URL of VSPs is encapsulated in a WS-Addressing element. A custom TNC specific element is used for encapsulating the name of a VSP, because

[21]The XML Schema of the proposed format can be found in section D.3.

WS-Addressing does not provide an appropriate element for this. As the list of VSPs is nested in a <*wsp:ExacltyOne*> statement (line 7), a client can choose one of the VSPs for performing the TNC check in order to comply with the policy requirements.

Alternatively, a client can also use implicitly trusted VSPs. The block with implicitly trusted VSPs is encapsulated with WS-SecurityPolicy elements (line 17) that indicate that the following policy statements relate to the digital signature of a security token. Furthermore, line 18 states that the X.509 certificate that contains the public key for verifying the signature shall be included in the signature[22]. This mechanism allows an SP to request certificates that are not stored in its local certificate store. An SP must be in possession of a VSP's certificate in order to validate the signature and the VSP's identity.

Lines 19-22 present a choice of two issuers of certificates that are accepted by an SP. Each issuer, that is, each CA, is identified by its *common name (CN)*. An SP tries to match the CNs of trusted CAs against the certificate that contains the signing key used to create the signature of the security token.

Both the integrity policy and the VSP policy are sent to the client in the TNC check request, that is, using IF-PAA. The following section describes how the policies are integrated into the XHTML-based TNC check request.

6.3.4 Combining XHTML, WS-Trust, and Policies

In the previous sections, three mechanisms have been described that, when combined, fulfil the requirements of IF-PAA. The XHTML-based approach can be used to embed a web-based TNC request into a Web page. This approach requires a mechanism that allows the SP to issue an authentication token. Such an approach for issuing tokens in browser environments was described in section 6.3.2. Finally, policy formats have been developed in section 6.3.3 that allow an SP to state its integrity and VSP requirements. This section combines these approaches and proposes a message format for IF-PAA.

Listing 6.5 shows a simplified example that outlines the structure of the proposed TNC check request format[23]. The entire request is enclosed by an HTML form (line 1). This form is used to send to the SP the security token that the user obtains from the VSP. The TNC request starts in line 2 with the element <*wtnc:TNCIntegrityCheckRequest*>. It is followed by an integrity policy in lines 3–11 and a VSP policy in lines 12-20. The optional security token that enables an authentication of the client at the VSP can be included in lines 21–23. As described in section 6.3.2, the security token is encapsulated in a WS-Trust RSTR message.

When the form is submitted, for example by a user clicking the button encoded in line 25,

[22] Cf. section 4.3.2.2.
[23] The XML Schema of the proposed format can be found in section D.4.

a browser plug-in intervenes in the submission and starts an external application that performs a TNC check with a VSP.

After the TNC check has been performed successfully, the client is in possession of a security token stating its compliance with the integrity policy[24]. The browser plug-in places this token into the HTML form that encloses the TNC check request and submits the form. As a result, the token is transferred to the SP using the HTTP POST method, as shown in listing 6.6. The value of the id parameter (line 2 in listing 6.5) is used as the name for the HTTP POST parameter for the security token (tncResultToken). The SP extracts the token from the HTTP request body and uses it for its authorisation decisions.

```
1  <form method="post" name="tncRequest" id="tncRequest"
       action="TNCCheckResult.aspx">
2  <wtnc:TNCIntegrityCheckRequest id="tncResultToken">
3   <wtip:IntegrityPolicy>
4    <wsp:Policy>
5     <wsp:ExactlyOne>
6      <wsp:All>
7       ...
8      </wsp:All>
9     </wsp:ExactlyOne>
10   </wsp:Policy>
11  </wtip:IntegrityPolicy>
12  <wtvp:VSPPolicy>
13   <wsp:Policy>
14    <wsp:ExactlyOne>
15     <wsp:All>
16      ...
17     </wsp:All>
18    </wsp:ExactlyOne>
19   </wsp:Policy>
20  </wtvp:VSPPolicy>
21  <wtnc:SecurityToken>
22   <wst:RequestSecurityTokenResponse> ... </wst:RequestSecurityTokenResponse>
23  </wtnc:SecurityToken>
24 </wtnc:TNCIntegrityCheckRequest>
25 <input type="submit" value="Start TNC check" />
26 </form>
```

Listing 6.5: Proposed format for requesting a TNC check

```
POST /TNCCheckResult.aspx
HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: localhost
tncResultToken=<saml:Assertion ...
```

Listing 6.6: Simplified example of a HTTP POST message transferring a security token to an SP

[24]Cf. section 4.4.

6.4 Summary and Conclusion

In this chapter, a mechanism for realising the communication between client and SP, that is, IF-PAA, was developed. IF-PAA is constrained to mechanisms that are supported by Web browsers. This reduces the choice of potential mechanisms to those that are based on TLS or HTTP.

Section 6.2 analysed the TLS-based authorisation extension TLS Authz. As pointed out in section 6.2.2, there are two issues associated with TLS Authz when considering it for IF-PAA. Firstly, TLS Authz is not flexible enough to allow all artefacts to be exchanged during a TNC integrity check request. Secondly, triggering the TNC check at the TLS level prevents authentication mechanisms on the application layer from being usable.

Consequently, section 6.3 focussed on a TNC check request mechanism that is based on application level mechanisms that do not have such limitations. HTTP and HTML are the dominant mechanisms supported by browsers at the application level. In order to reuse existing browser functionality (such as HTML forms), an approach based on embedding the TNC check into an HTML document was proposed in section 6.3.1.

The TNC check needs to be performed by a client application that queries IMC components and reports the measurement to the VSP, as described in chapter 5. The idea of the approach described in section 6.3.1 is to trigger this client application from an HTML document. Object tags (described in section 6.3.1.1) cannot be used reliably, as they are subject to being disabled by browser security settings. An alternative approach based on XHTML and outlined in section 6.3.1.2 is unaffected by browser settings. Furthermore, because this approach is XML-based, it provides a more flexible way of expressing a TNC check request.

The description of Cardspace in section 6.3.1.3 exemplified the use of an XHTML-based approach. While the Cardspace XHTML syntax cannot be used to trigger a TNC check, the basic idea of using XHTML to trigger a client application is applicable to TNC.

Section 6.3.2 provides an encapsulation for authentication tokens that are issued by SPs. The WS-Trust RSTR message format is reused and combined with the XHMTL-based approach.

In order to state its integrity and VSP requirements, the SP needs a customised policy format. Section 6.3.3 proposed such policy formats that are based on the standardised policy framework WS-Policy.

All three approaches, that is, the XHTML message format, the WS-Trust-based token issuance, and the policy formats, are combined in section 6.3.4. The result is a mechanism that is based on open standards that can be used to trigger a TNC check using a browser and to return the TNC check result to the SP. As such, it fulfils the requirements of IF-PAA, as stated in section 6.1. The previous chapters introduced a communication model and a set of interfaces that can be used to realise a web-based TNC check. The proposed message formats and mechanisms have been combined in a prototype implementation. A description of this implementation is given in the next chapter.

Chapter 7

Implementation and Evaluation

The previous chapters proposed the building blocks that can be combined to enable a web-based TNC check. These building blocks, that is, an architecture, message and policy formats, as well as protocols have been combined into a prototype implementation. This chapter summarises the implementation of this prototype and presents the results of a test bed evaluation.

Figure 7.1 gives an overview of the implemented components and their interfaces. The client, as described by the assertion-based architecture, performs a TNC check with a VSP. Both, client and VSP, use a common TNC library, which realises the interface IF-TNCCS. This library is introduced in section 7.1. A web-based TNC check is triggered in a browser environment using the interface IF-PAA. Section 7.2.1 describes the design and implementation of a browser plug-in that enables this behaviour. Furthermore, a client desktop application has been developed for this project that performs the role of the Network Access Requestor, as defined in the TNC specification. This application is named *wTNC client* and is described in section 7.2.2. In addition, a helper application has been developed to assist the *Integrity Measurement Layer* in collecting measurements. Its implementation is summarised in section 7.2.3. Measurements, encapsulated using the mechanisms defined by IF-T, are sent to a VSP. Implementation details about the VSP are given in section 7.3. The TNC check results are encapsulated in a security token and sent to an SP. Section 7.4 describes how the SP was implemented. All components have been combined in a test bed, where performance tests were conducted. Section 7.5 describes the test bed and summarises the test results. Finally, section 7.6 concludes this chapter.

7.1 Reusing TNC Implementations

As previously discussed, a requirement of the web-based TNC check is to keep the interfaces in the upper layers of TNC (IF-M and IF-TNCCS) unchanged in order to reuse existing TNC software. This approach enables the reuse of existing TNC software stack for this prototype implementation. A side effect of reusing an existing software stack is that it can be used to confirm that the implemented prototype conforms with the existing TNC interfaces.

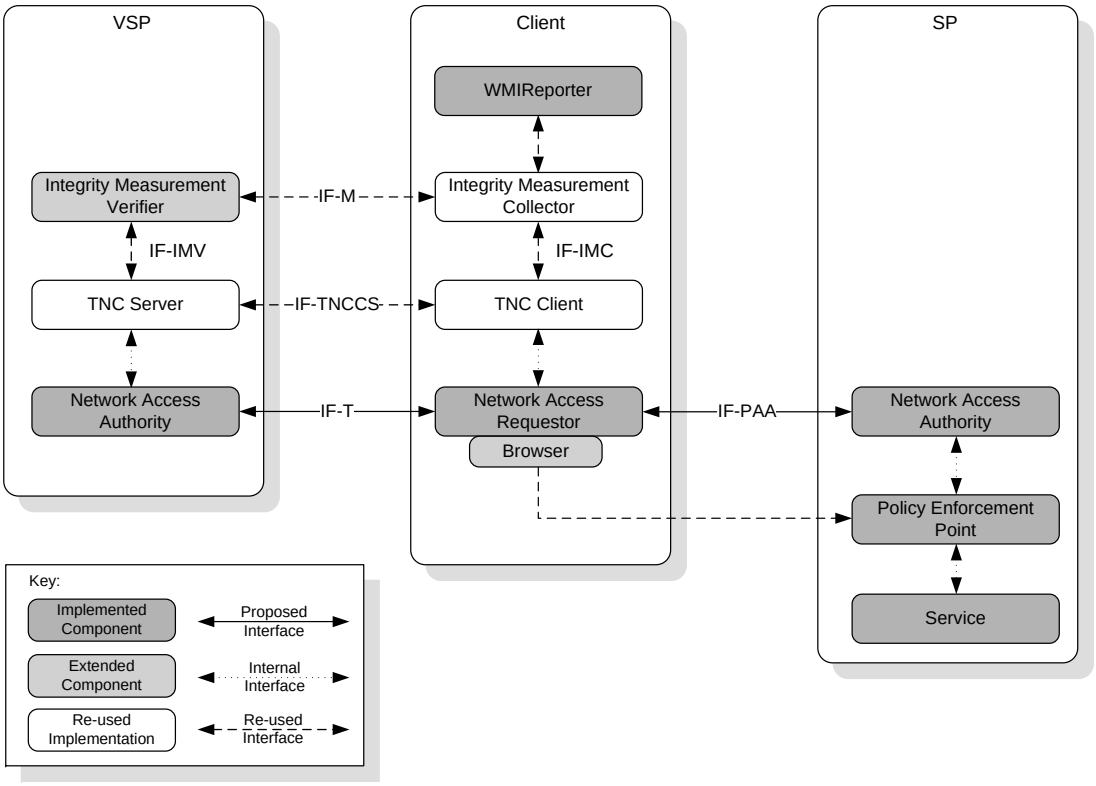


Figure 7.1: Overview of the implemented entities and interfaces

The Trusted Computing Group lists two open source projects that implement TNC[1]: *libTNC* and *TNC@FHH*.

TNC@FHH[2] is a C++ TNC implementation created at the University of Applied Sciences in Hanover, Germany. It features a Windows-based access requestor and a TNC server implemented as a FreeRadius[3] module for Linux operating systems. The TNC client, however, is no longer maintained and its development is discontinued. The latest version of the TNC client software is not fully usable, as it is limited to four TNC messages that can be exchanged during one TNC check.

libTNC[4] is a library that covers loading and communicating with IMVs and IMCs, a sample IMC/IMV pair, and support for creating XML-based TNCCS-Batches. It is written in C and has been tested on Windows, Linux and MacOS. In contrast to TNC@FHH, libTNC is not a complete TNC solution. It covers only the upper layers of the TNC architecture while leaving other areas, such as, for example, network communication and graphical user interface, out of its scope.

[1]See https://www.trustedcomputinggroup.org/groups/network/.
[2]See http://tnc.inform.fh-hannover.de/wiki/index.php/Main_Page.
[3]FreeRadius is an open source RADIUS server (http://www.freeradius.org).
[4]See http://sourceforge.net/projects/libtnc for the project web page. The latest version is available from http://linux.softpedia.com/get/Programming/Libraries/libtnc-37575.shtml.

In contrast to TNC@FHH, libTNC is written with the intent of being used as part of a program. Furthermore, the EAP-based message exchange[5] is hard-wired into TNC@FHH and it is further bound to the FreeRadius server. Because of its openness and focussed scope, libTNC has been chosen to be used in this project.

While it is important to find a library that provides TNC functionality, it is equally important that the prototype implementation is not bound to a particular library. The architecture of all affected components needs to be designed in a way that allows for exchanging the underlying TNC library without requiring changes in other parts of the application. Architectural decisions related to the integration of libTNC are discussed in sections 7.2.2.2 and 7.3.1. In the following, the client part of the solution that has been implemented in this project is described.

7.2 Client Applications

The client implementation consists of two core parts: a browser plug-in that communicates with the SP and a client desktop application that performs the TNC check. Furthermore, a third component has been implemented for this project that acts as a helper component for the IMC layer and collects integrity measurement data. The design and implementation of these components are described in the following.

7.2.1 wTNC Web Browser Extension

In the web-based TNC communication model[6], a client accesses an SP which in turn requests a TNC check. The interface between client and SP has been proposed in chapter 6. As described in section 6.3, an SP encodes the TNC request in an XHTML page. A standard browser is unaware of the semantics of the TNC request. The ability to handle such a TNC request can be added through a browser plug-in.

For this project, a plug-in for the Web browser Mozilla Firefox[7] has been developed. As plug-ins in Firefox are called *extensions*, it is named *wTNC browser extension*. Firefox extensions are packed and distributed as compressed zip files with an *.xpi* file extension. Typically, such an XPI file is downloaded from an online repository[8] and then installed[9].

The user interface of a Firefox extension is described using the *XML User Interface Language (XUL)*[10] language. The logic behind the user interface can be implemented in either C++ or

[5] Cf. section 2.3.1.
[6] As proposed in section 4.4.
[7] See http://www.mozilla.com/firefox/.
[8] Such as the official Firefox add-on Web site https://addons.mozilla.org/firefox/.
[9] See section A.1.1 for instruction on how to install the wTNC extension.
[10] See http://www.xulplanet.com/ and http://developer.mozilla.org/en/XUL for more information.

Javascript. For this project Javascript was chosen, as it allows to test functionality without recompiling the extension. It is further possible to debug extensions written in Javascript directly in the browser.

In order to perform a TNC integrity check, a user does not have to interact with the wTNC browser extension. The extension works in the background and only requires a user interface for configuration purposes. Figure 7.2 shows a screenshot of the configuration interface. As previously described, the browser extension launches a desktop application that performs the TNC check. The first configuration setting specifies where this application can be found. Using the second setting, detailed logging can be enabled.

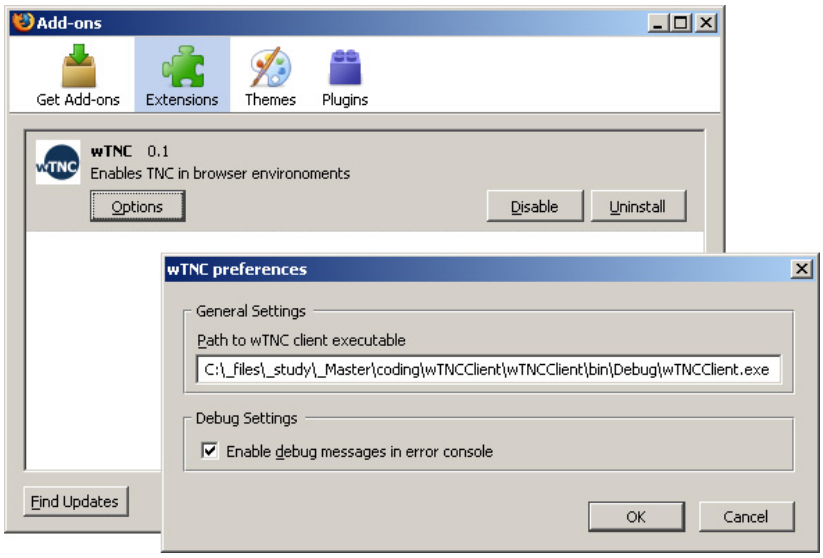


Figure 7.2: Configuring the wTNC browser extension for Firefox

The tasks performed by the wTNC browser extension can be grouped into four phases.

Phase 1 "Hook" into the browser for receiving notifications when a web page is loaded.

Phase 2 Detect a TNC check request and hook into the related HTML form.

Phase 3 Intercept submission of the HTML form and start desktop application for performing the TNC check.

Phase 4 Retrieve TNC result (in form of a security token) and insert it into the HTML form.

The wTNC browser extension that has been developed for this project consists of three classes (*wTNCControl*, *ProgressListener*, and *AppLauncher*). Figure 7.3 summarises their interaction in a UML[11] sequence diagram.

[11]The Unified Modeling Language (UML) is a standardised language that defines 13 diagram types for visualising aspects of a software system. See http://www.uml.org for more information.

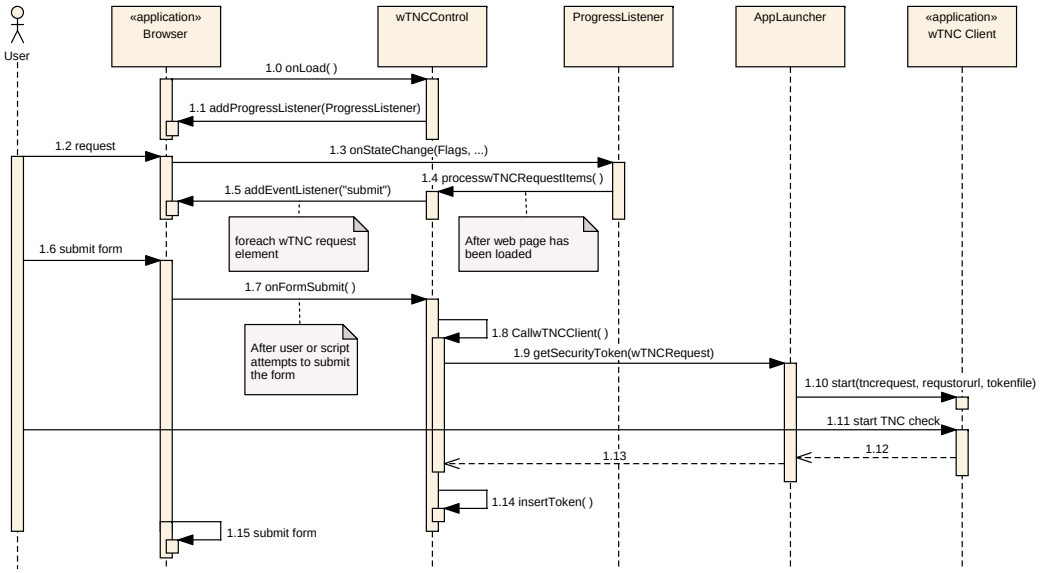


Figure 7.3: UML sequence diagram showing the implemented communication flow of the wTNC browser extension

Phase 1 The wTNC browser extension is initialised during the Web browser start up process. As described above, this extension monitors if a new Web page has been loaded before it checks whether this Web page contains a TNC check request. The monitoring is realised using an event handler approach. An event handler has been implemented (*ProgressListener*) that is registered in the *onLoad* method (message 1.0) during the browser start up process. This event handler receives all events that are related to changes in the progress of loading a web page (message 1.1) from the browser, that is, every time the browser's progress state changes, the *ProgressListener* will be informed.

Phase 2 Before this phase can be reached, a user has to instruct the browser to load a Web page (message 1.2). After the browser has informed the event handler that a new Web page has been loaded (message 1.3), the event handler checks whether this Web page contains a TNC check request. As described in section 6.3.4, such a request is enclosed by a *<wTNC:IntegrityCheckRequest>* element.

While loading a Web page, a browser creates a tree that contains all elements that are contained within that page (for example XHTML elements, images, and links). This tree can be accessed using the *Document Object Model (DOM)*[12], a standard API for accessing and manipulating XML-based data. Using the DOM API, the wTNC browser extension checks whether the document tree contains an *IntegrityCheckRequest* element (message 1.4). If this element could not be found, the wTNC browser extension stops processing the current document and waits for the next Web page to be loaded by the browser.

[12]More information about the DOM can be obtained from http://www.w3.org/DOM/.

If one or more *<wTNC:IntegrityCheckRequest>* elements have been found, the extension will further check whether these elements are enclosed by an HTML form element. At a later stage, this form will be used to send the security token that contains the TNC check result to the SP. If an enclosing form is found, an event listener (*onFormSubmit*) is registered that is notified by the browser when this form is submitted to the SP (message 1.5).

Phase 3 Before this phase is reached, a user (or a script acting on the user's behalf) must have invoked the submission of the HTML form that contains the TNC check request (message 1.6). Before sending the contents of the form to the SP, the browser executes the event handler that has been registered in the previous step (*onFormSubmit* in message 1.7). The *onFormSubmit* method invokes the *CallwTNCClient* method, which transforms the DOM representation of the TNC check request into a string (message 1.8). This string, which contains the policies as expressed by the SP, is sent to the *AppLauncher* class (message 1.9). This class is responsible for launching the wTNC client application[13].

The wTNC client is implemented as a standalone desktop application. In order to allow this application to communicate with the wTNC browser extension, an interprocess communication (IPC) mechanism was implemented. This mechanism allows the client application to receive the TNC check request parameters and to return the TNC check result to the browser extensions. The TNC request parameters are sent to the wTNC client by using command line arguments. The security token that contains the TNC check result is stored in a temporary file. The wTNC client is launched with three command line arguments (message 1.10). These arguments include the TNC check request as sent by the SP, the URL of the Web page that requested the TNC check, and the path to a temporary file created by the browser extension in which the security token is expected.

Phase 4 The final phase is initiated after the TNC check has been performed (message 1.11). The wTNC client application is started in a *blocking* mode, that is, the execution of the browser extension is suspended until the wTNC client application terminates. After the TNC integrity check has been performed, the wTNC client application stores the resulting security token in the temporary file and terminates. This resumes the execution of the browser extension, which reads the content of the temporary file before deleting it (messages 1.12 and 1.13).

The security token needs to be placed into the HTML form, as only data that is contained in HTML form fields (such as a text field, a check box, or a drop-down box) is sent to the form receiver. The wTNC browser extension therefore creates a hidden form field and places the string representation of the security token into this field (message 1.14).

[13]This class is partly based on the Bandit project. See http://code.bandit-project.org/trac/wiki/DigitalMe.

After the retrieved security token has been placed into the form, the execution of the wTNC browser extension is completed. By handing the control over to the browser, the form is submitted and the token is sent to the SP (message 1.15).

The wTNC browser extension enables a TNC integrity check in a web-based environment. That is, it bridges between an XHTML based communication on one side and a SOAP-based communication on the other side. The TNC integrity check itself is performed by a standalone application. Implementation details about this standalone application are given in the following section.

7.2.2 wTNC Client Application

The wTNC client application *(wTNC client)* provides the user interface for the TNC integrity check. This application, which has been designed and implemented for this project, has two main responsibilities. Firstly, it is responsible for performing the TNC check with a VSP. Secondly, it is also used for managing the list of VSPs that are trusted by the user. The user interface has been kept simple, featuring only essential control elements, as the screenshot in figure 7.4 shows.

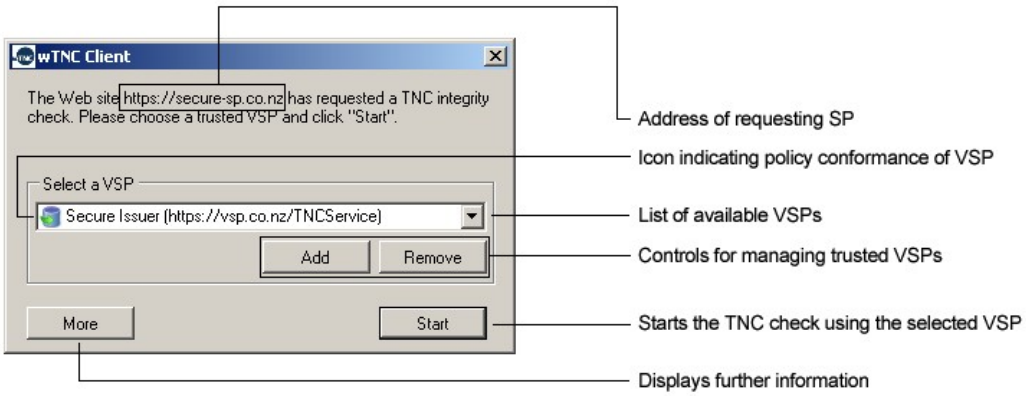


Figure 7.4: The wTNC client and its user controls

The wTNC client has been implemented using C# and the .Net framework 3.5[14]. It is a standalone application that is started by the wTNC browser extension. It indicates the URL of the requesting SP to the user. Based on this URL, the user can decide whether the TNC check result shall be released to this SP.

The user must also decide which VSP shall be utilised for performing the TNC check. A list of all available VSPs is shown in a drop down menu. This list contains all VSPs that are

[14]C# and the .Net framework were chosen for this project, as they provide integrated support for communication mechanisms, such as SOAP over HTTP. For other language environments, such as C or C++, this functionality is provided only by external libraries. The integration of such libraries would have resulted in additional development time.

proposed by the SP using its VSP policy[15] and those that are trusted by a user. Icons are used to indicate whether a VSP is trusted by a SP and/or the user. The following section provides more detail about this labelling and how the list of trusted VSPs can be managed.

Furthermore, section 7.2.2.2 describes how libTNC was integrated into the wTNC client. As proposed in chapter 5, a WS-Trust based communication scheme is used for transporting TNC messages. Section 7.2.2.3 provides an architecture-focussed description of how TNC and WS-Trust have been combined. Finally, section 7.2.2.4 gives an overview about how the wTNC client performs a TNC check.

7.2.2.1 Managing VSPs and VSP Policies

In addition to performing the TNC check, the wTNC client can also be used to manage the list of VSPs that are trusted by the user. This list reflects the user's policy and defines which VSPs are trusted for performing a TNC integrity check. The wTNC client supports two methods for adding a new trusted VSP. The first option is to manually add a VSP. The wTNC prompts the user for the VSP details, such as name, URL, and supported CAs. As this method can be error prone, it is further possible to add a VSP automatically through a browser. Similar to a TNC check request, an XML fragment can be placed into a Web site that contains the VSP details. This fragment is detected by the wTNC browser extension and forwarded to the wTNC client. Before a new VSP is added to the list of trusted VSPs, a user checks the details of the VSP and confirms the process.

A further trust relation exists between SP and VSP, as an SP might only trust certain VSPs. As described in section 6.3.3.3, an SP can express this trust relation using a policy that is part of the TNC request. The wTNC client interprets this policy and combines it with the user's VSP policy. The result of this process is a list of all available VSPs in which each VSP is associated with a status icon. This icon reflects the trust relation between SP and VSP as well as the trust relation between client and VSP. Based on this status icon, a user can choose a VSP that he or she trusts and from which the SP will accept a TNC check result. Figure 7.5 shows the status icons that have been created for the wTNC client.

A VSP policy sent by an SP can contain both explicit and implicit information for expressing trust relations with VSPs. All VSPs that are explicitly trusted by an SP (by stating their URLs) are displayed using icon a). If an explicitly trusted VSP is also contained in the user's policy, that is, it is mutually trusted, icon b) is used instead. A green tick indicates that this VSP is mutually trusted.

The same tick is also used in icon c) to indicate that a VSP that is trusted by the user supports a CA that is trusted by the SP (implicit trust relation). For all other VSPs that are trusted by the user but not compliant with the VSP policy, icon d) is used. The warning sign in this

[15] Cf. section 6.3.3.3.

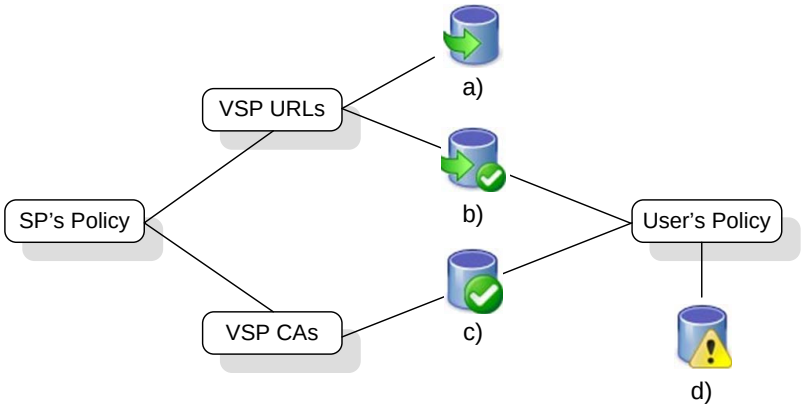


Figure 7.5: Icons used in the wTNC client to visualise a VSP status

icon indicates that, according to its policy, the SP will not accept a TNC check result signed by this VSP.

Using this icon-based mechanism, a user can pick a VSP that is mutually trusted for performing the TNC integrity check. After a VSP has been selected, the TNC check can be performed with libTNC as the underlying TNC library. The integration of libTNC into the wTNC client is described next.

7.2.2.2 Integration of libTNC

The wTNC client has been implemented in C# using the Windows Communication Foundation (WCF). WCF is a framework that, amongst other communication schemes, provides support for exchanging messages using SOAP-based Web services in the .Net framework [Low07, RCB08]. As such, WCF provides the basic communication layer for the wTNC client.

The upper layers in the TNC architecture have been retained for the web-based TNC check which allows the reuse of existing TNC software, such as libTNC. This library is used by the wTNC client for loading IMCs and for encapsulating the communication between IMCs and IMVs in TNCCS-Batch messages. LibTNC provides an API written in C, which needs to be invoked in an appropriate manner. This requires a mechanism that allows C code to communicate with C# code and vice versa. This interoperability between C and C# is not provided by .Net. A customised solution was implemented for this project that circumvents this limitation. It is described in the following.

As mentioned in section 7.1, one design goal for the wTNC client is to allow exchanging the underlying TNC library without having to change other parts of the application. At the same time, the TNC library must not be aware of the underlying transport mechanism. Otherwise every TNC library that is used within the wTNC client would need to be modified

so that it "speaks" WS-Trust. To prevent this, a layered approach has been designed and implemented for this project that allows libTNC's C API and a WS-Trust based communication to be used together without coupling them tightly. Table 7.1 shows these layers and their responsibilities. Furthermore, the table shows the names of the components (for example, classes) that are implemented at a particular layer. These components are depicted in figure 7.6, where a UML class diagram shows their static relation[16].

Table 7.1: Overview of layers and their responsibility in the wTNC client architecture

Layer	**Component(s)**	**Responsibility**
TNC Implementation Layer (TIL)	libTNC	Provides an interface to IMCs and the TNC client.
TNC Implementation Encapsulation Layer (TIEL)	libTNCClientDLLWrapper	Provides a TNC implementation dependent interface.
TNC Implementation Abstraction Layer (TIAL)	LibTNCClientWrapper	Provides a TNC implementation independent interface for performing a TNC check.
Communication Abstraction Layer (CAL)	ClientWSStack	Abstracts the underlying communication methods.
Communication Implementation Layer (CIL)	WSTrustServiceClient and WCF	Provides support for sending and receiving SOAP messages.

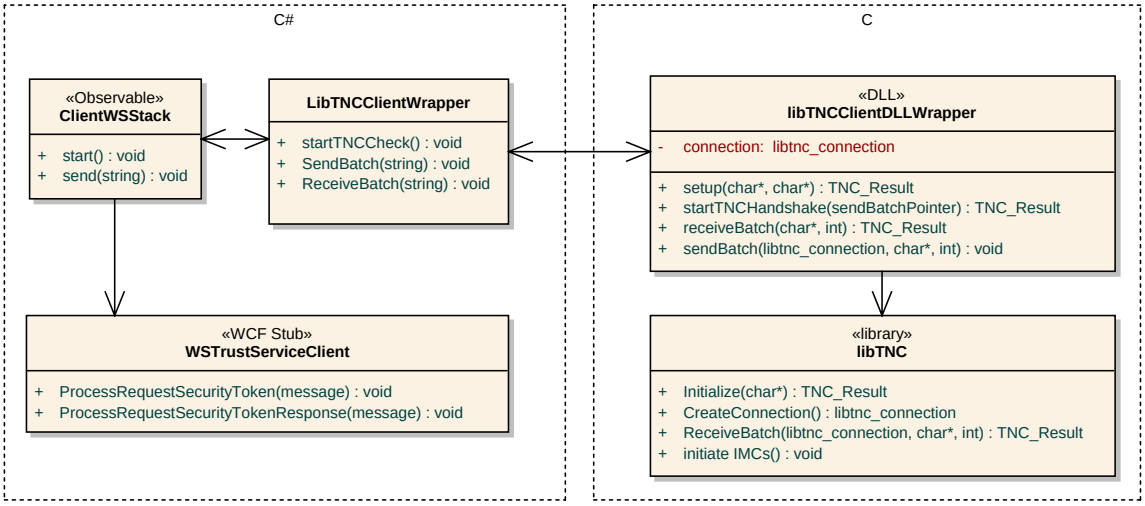


Figure 7.6: Simplified class diagram showing the encapsulation of libTNC and the language boundary between the C and the C#

The top layer, as shown in table 7.1, is the *TNC Implementation Layer*. At this layer, libTNC is located and provides core TNC functionality. An excerpt of libTNC's API is represented using the UML class notation (in the lower right corner in figure 7.6)[17].

[16] This class diagram shows only a subset of the classes used to implement the wTNC client. See figure B.1 for a complete class diagram.

[17] In favour of clarity, the function names have been shortened for this description. In libTNC, all functions related to the client are prefixed with *libtnc_tncc_*.

The communication with the *TNC Implementation Layer* is based on TNCCS-Batches. In order to pass a TNCCS-Batch *into* libTNC (and therefore to an IMC), libTNC provides the *ReceiveBatch* function. After a TNCCS-Batch has been processed, a reply message (also in form of a TNCCS-Batch) is created by libTNC. Because libTNC does not have its own communication layer, it requires the surrounding application, in this case the wTNC client, to provide a *SendBatch* method that is capable of sending a TNCCS-Batch to the TNC Server.

However, because libTNC is a C library and the wTNC client is written in C#, they cannot interact directly with each other[18]. To overcome this limitation, libTNC is wrapped into a DLL[19]. As C# can access functions in DLLs, this approach bridges between C# code and libTNC. This DLL is implemented at the *TNC Implementation Encapsulation Layer*. In the class diagram, it is represented as the *libTNCClientDLLWrapper* class. The DLL further stores libTNC's connection identifier of the current TNC check, that is, it encapsulates the internal connection handling of libTNC.

Wrapped in a DLL, libTNC can be accessed from C#, that is, C# can call functions written in C. The class responsible for communicating with the DLL is *LibTNCClientWrapper*, which is implemented at the *TNC Implementation Abstraction Layer*. This class has been designed with a simple interface for performing a TNC check that hides the API of libTNC. By using this abstraction mechanism[20], it is possible to replace libTNC with a different TNC library without changing other parts of the wTNC client application.

As mentioned above, libTNC expects a *SendBatch* function in the surrounding application code. Wrapping libTNC in a DLL allows calling C functions from a C# environment. However, this approach does not allow a communication in the other direction, that is allowing C code to access C#. In order to bridge the communication from C to C#, a callback mechanism was implemented that is based on an indirection. When a TNC check is initiated, a pointer to the *SendBatch* implementation in the *LibTNCClientWrapper* object (written in C#) is passed to the DLL (*libTNCClientDLLWrapper*). The DLL stores this pointer and forwards calls to its *SendBatch* function to the *SendBatch* method of the *LibTNCClientWrapper* object. This mechanism allows the libTNC C library to indirectly call C# methods.

The *LibTNCClientWrapper* itself does not know how to send a TNCCS-Batch using WS-Trust. Instead, this knowledge is encapsulated at the *Communication Abstraction Layer* in the class *ClientWSStack*. This class provides a simple API that allows the *LibTNCClientWrapper* to send TNCCS-Batches wrapped in WS-Trust messages. The ClientWSStack uses the *WSTrustServiceClient* class (which in turn uses WCF) on the *Communication Implementation Layer* to send messages to the TNC Server (located at the VSP). The reply to such a

[18] In .Net terms, it is not possible to call *managed* C# code from *unmanaged* code and vice versa [Gre03, page 101].
[19] *Dynamic Link Libraries (DLL)* are the Windows-specific format for creating shared libraries.
[20] Which is an application of the *Facade* design pattern [GHJV94].

message (in form of a TNCCS-Batch) is passed from the *Communication Implementation Layer* to libTNC in the *TNC Implementation Layer* using a chain of function calls.

The class *ClientWSStack* has been made exchangeable by applying the *Observer* design pattern[21]. This mechanism proved useful during the development of the wTNC client, where the *ClientWSStack* was replaced with a *ClientMockStack* class that allowed the wTNC client to communicate directly with the VSP without requiring a network. This mechanism can further be used to replace the underlying communication framework without changing other parts of the application.

In this section, a static excerpt of the architecture of the wTNC client was given. As described above, the architecture is flexible and allows exchanging the underlying TNC library as well as the underlying transport library. In the current implementation, WCF is used as the transport library. While WCF provides support for WS-Trust, it could not be used for creating and parsing WS-Trust messages, as described in the following section.

7.2.2.3 Integration of WS-Trust

WS-Trust is the protocol that underlies all communication between wTNC client and VSP. WCF provides an implementation of WS-Trust. However, this implementation does not fully support the WS-Trust specifications [Nad07]. In particular, two extension mechanisms in WS-Trust are not part of the WCF implementation, which are required for realising the TNCCS-Batch exchange as described in section 5.3.4.2. Firstly, the WCF implementation of WS-Trust does not support a message exchange that exceeds a simple request-response pattern. Secondly, WCF does also not support the *Negotiation and Challenge Extensions*, which allows embedding TNCCS-Batches into WS-Trust RST and RSTR messages[22].

Unfortunately, the internal implementation of WS-Trust in WCF is hidden and can therefore not be extended[23]. These limitations make it impossible to rely on the WCF implementation of WS-Trust for the purposes of a TNC check.

However, a collection of sample applications provided by Microsoft[24] contains a partially implemented WS-Trust stack. While this implementation lacks some core functionality, such as reading RSTR messages, it could still be reused. This prevented that the entire WS-Trust protocol had to be reimplemented. In addition to the missing core functionality, this

[21] The Observer pattern allows an object A to get notified if the state of object B changes. This prevents that object A needs to know the interface of object B and avoids an approach that is based on polling the state of object B [GHJV94].

[22] As proposed in chapter 5.

[23] While it was possible to extend the internal implementation of WS-Trust in a previous version of WCF, this possibility has been removed. See "Remove support for RST/RSTR processing" at http://wcf.netfx3.com/content/BreakingChangesbetweenVistaBeta2andJuneCTP.aspx#_Toc140563002.

[24] See *Windows Communication Foundation (WCF), Windows Workflow Foundation (WF) and Windows CardSpace Samples* at http://www.microsoft.com/downloadS/details.aspx?FamilyID=2611a6ff-fd2d-4f5b-a672-c002f1c09ccd&displaylang=en.

sample WS-Trust implementation does also not include support for the WS-Trust extensions mentioned above. However, because the source code is available, it was possible to add the missing functionality. The class diagram in figure 7.7 gives an overview of the resulting implementation.

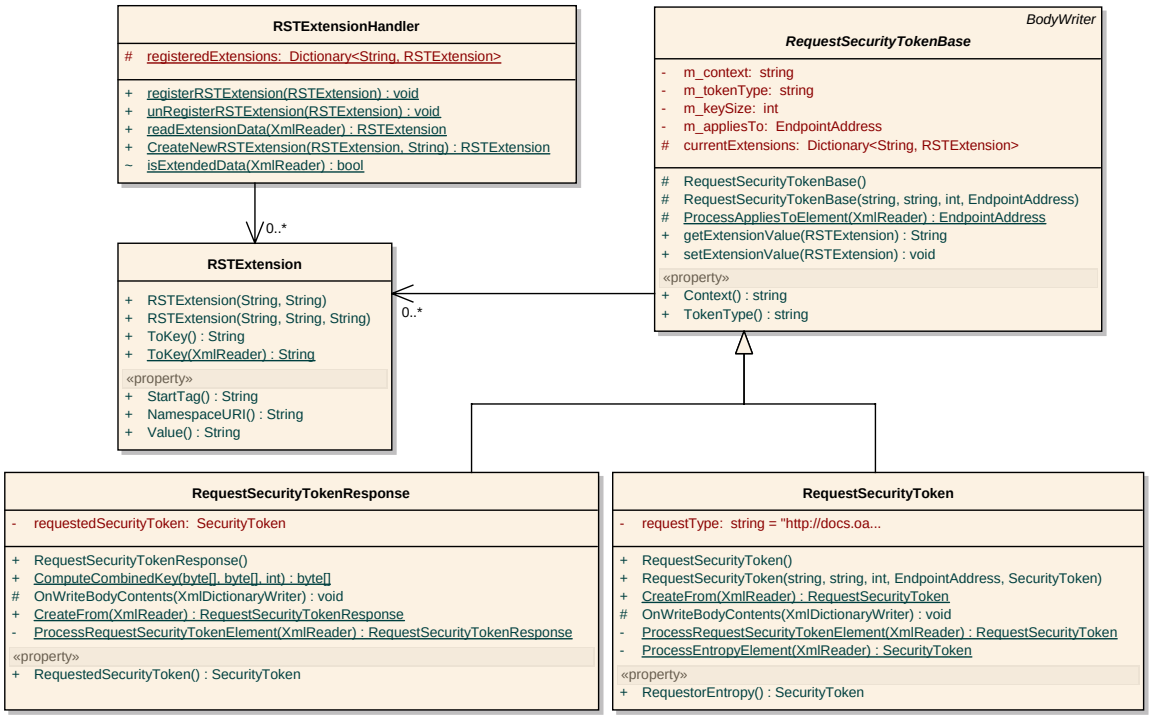


Figure 7.7: Classes involved in creating and parsing WS-Trust messages

The core classes for parsing and writing RST and RSTR messages are *RequestSecurityToken* and *RequestSecurityTokenRsponse*, respectively. These classes have a common abstract base class (*RequestSecurityTokenBase*) which implements the *BodyWriter* interface. By implementing this interface, objects of this and all derived classed can be sent as part of a SOAP message using WCF. The XML representation of an RST or RSTR object is retrieved by WCF by calling their *OnWriteBodyContents* method while assembling an outgoing SOAP message. RSTs and RSTRs can be loaded from their XML representation using their *CreateFrom(XMLReader)* method. These methods make it possible to create RST and RSTR messages without relying on WCF. This allows the arbitrary creation and exchange of RST and RSTR messages between wTNC client and VSP. This effectively enables the realisation of the first WS-Trust extension that is part of the proposed mechanism for exchanging integrity measurements, that is, a message exchange beyond a simple request-response pattern.

The second WS-Trust extension, which is responsible for including additional XML structures in an RST or RSTR message, was implemented using two helper classes. The class *RSTExtensionHandler* provides a static interface that can be used for registering WS-Trust extensions. Before an extension can be registered, it needs to be defined. The class *RSTEx-*

tension provides a data structure for this. Line 2 in listing 7.1 provides a code example of the developed API for registering a WS-Trust extension. After an extension has been registered, its value inside a RST or RSTR can be read (line 5) or a new value can be assigned (line 8). By using this extension mechanism, TNCCS-Batches and policy information can be attached to WS-Trust messages.

```
1 //Register a WS-Trust extension
2 private RSTExtension TNCCSExtension = new
      RSTExtension(Constants.TNCCS.TNCCSBatch, Constants.TNCCS.NamespaceUri);
3
4 //Reading the content of an extension
5 rst.getExtensionValue(TNCCSExtension);
6
7 //Assigning a value to a previously registered extension
8 rstr.setExtensionValue(RSTExtensionHandler.CreateNewRSTExtension(TNCCSExtension,
      msg));
```

Listing 7.1: Code samples showing how to register, read, and write to and from a WS-Trust extension

7.2.2.4 Performing a TNC Check with the wTNC Client

The previous sections described various aspects of the wTNC client. This section combines these aspects and shows how the wTNC client performs a TNC check. The UML sequence diagram in figure 7.8 provides an overview of the implemented classes and their interaction.

The TNC check is triggered by a user selecting the *Start* button on the wTNC client user interface as shown in figure 7.4. User interaction is handled by the class *MainForm*, which forwards the user's request to start the TNC check to the *ApplicationHandler* (message 1.1). This class links the user interface with the components that performs the TNC check. A new *ClientWSStack* object is instantiated and supplied with VSP connection details and the policies that have been obtained from the SP through the browser extension (message 1.2).

Following this step, the *ApplicationHandler* registers as an observer of the *ClientWSStack* object (message 1.3). This mechanism enables the *ClientWSStack* to monitor the TNCCS-Batch exchange and to inform the *ApplicationHandler* if an access decision (issued by the VSP) has been made (message 1.19).

As described in section 7.2.2.2, the libTNC library is encapsulated by the *LibTNCClientWrapper* class. It has been designed to abstract the TNC layer from the rest of the application and thus making it possible to change the underlying TNC library. This class is informed about the TNC check via the *ClientWSStack* (messages 1.4 and 1.5). After setting up all configuration data in message 1.6, the callback pointer to the *sendBatch* method in the *LibTNCClientWrapper* is passed to the DLL (message 1.7). The code in the DLL sets up

libTNC (message 1.8)[25] and starts the TNC check with message 1.10. Internally, libTNC informs all installed IMCs that a TNC integrity shall be performed (message 1.9). As a result, these IMCs perform initial measurements and send the result to libTNC where the results are encapsulated in a TNCCS-Batch.

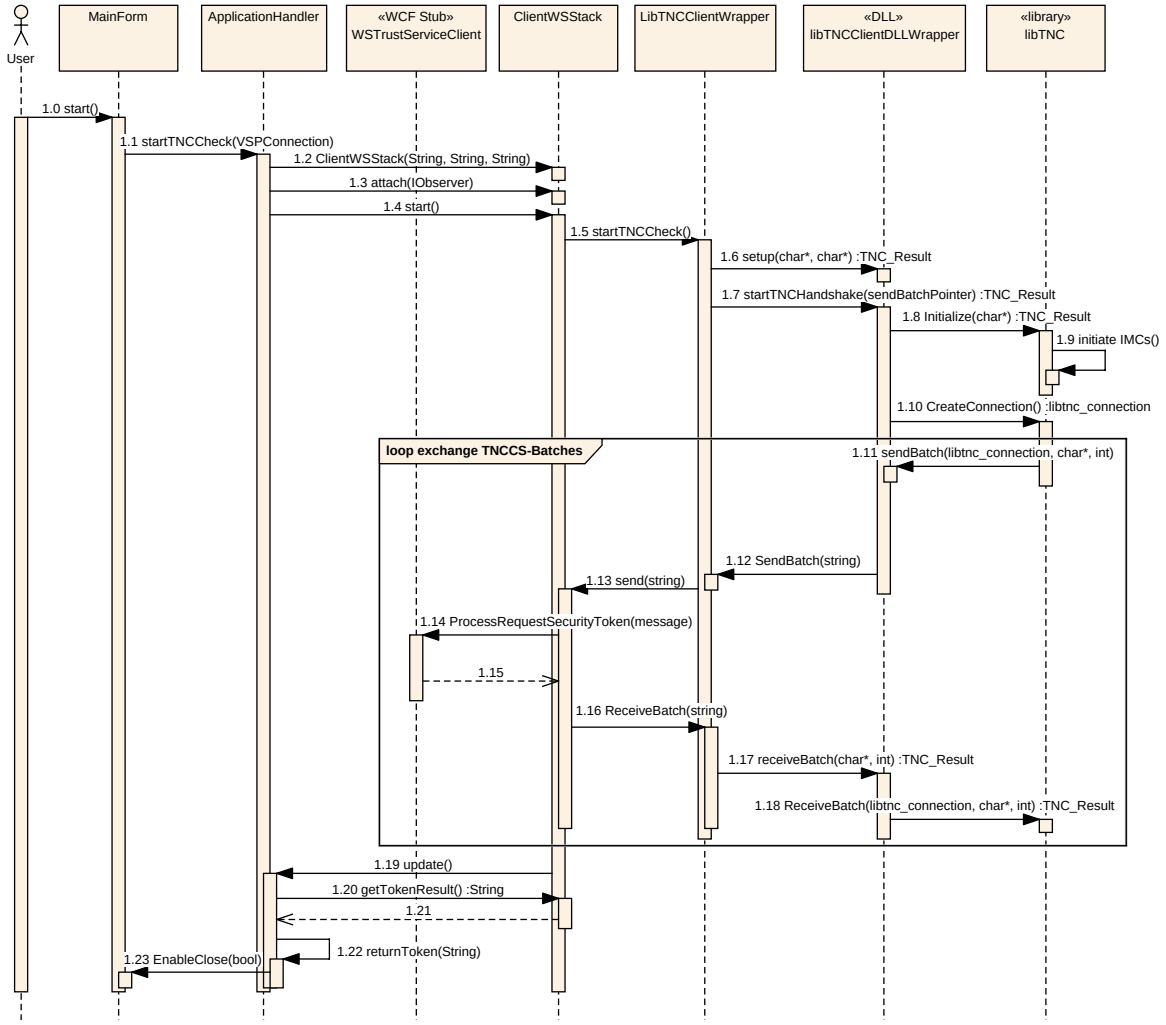


Figure 7.8: Sequence diagram showing the TNC message exchange

This TNCCS-Batch needs to be sent to the VSP. LibTNC invokes the *sendBatch* method provided by the DLL (method 1.11), which in turn, calls the *sendBatch* method provided by the *LibTNCClientWrapper* (method 1.12). The TNCCS-Batch is now handled in a C# environment. Message 1.13 sends the TNCCS-Batch to the *ClientWSStack* where it is prepared to be sent to the VSP using WS-Trust. The first message in a WS-Trust message exchange will be encapsulated in an RST message. All subsequent messaged use the RSTR message format[26]. Depending on the message type, the *ClientWSStack* invokes the *ProcessRequestSecurityToken* (for RST messages) or the *ProcessRequestSecurityTokenRequest* method (for RSTR messages)

[25] Additional setup messages are being exchanged. Their description has been omitted in favour of clarity.
[26] Cf. section 5.3.4.

of the WCF client stub implementation (*WSTrustServiceClient*) in message 1.14. Consecutively, WCF uses the custom WS-Trust implementation described in the previous section to retrieve the XML representation of the WS-Trust messages, which is sent to the VSP. The VSP processes the message and responds with a WS-Trust RSTR message (message 1.15). The *ClientWSStack* extracts the TNCSS-Batch contained in the RSTR message and passes it on to libTNC in messages 1.16 to 1.18, where the contents are evaluated by libTNC and its registered IMCs.

The message exchange from message 1.11 to message 1.18 will be repeated until the TNC check has been finished, that is, until an access recommendation has been made and no further TNCCS-Batches need to be exchanged. As mentioned before, the *ClientWSStack* monitors whether a WS-Trust RSTR message contains a SAML security token. As described in section 4.3.3, this SAML token contains the access recommendation of the VSP. If such a token is found, the previously registered *ApplicationHandler* will be informed (message 1.19). In turn, the *ApplicationHandler* will request the token from the *ClientWSStack* (message 1.20). The returned token (message 1.21) is stored in a temporary file as specified by the wTNC browser extension (message 1.22). Finally, the *ApplicationHandler* causes the *MainForm* to display a message to the user, indicating that the TNC check has been performed successfully, and causes the wTNC client to be terminated. As described in section 7.2.1, the wTNC browser extension resumes and obtains the token from the file before forwarding it to the SP.

A helper application has been implemented for this project that is used to gather integrity measurement data. This application is briefly described in the following section.

7.2.3 WMIReporter

This thesis focusses on the base mechanism that allows the TNC IMC/IMV layer to exchange measurements in a web-based environment. In order to test this message exchange in a realistic environment, an IMC/IMV pair is necessary that can report and evaluate different aspects of a system. The libTNC library includes an IMC/IMV pair that can be used for testing purposes. This IMC, called *OSC-IMC*, can report basic measurements about the operating system (for example version and installed service packs), files and their attributes stored on the client, and, on Windows operating systems, the values of keys in the registry [McC08]. The OSC-IMV gathers this information through the operating system's API. In order to extend the list of possible measurements, the OSC-IMV supports the execution of an external application that performs and reports further measurements.

For this project, such an external helper application, called *WMIReporter*, has been implemented. It extends the measurement capabilities of OSC-IMV and is briefly described in the following.

The WMIReporter is implemented as a command line application. It can provide information about personal firewalls and anti-virus software products that are installed on a Windows system. The purpose of the WMIReporter is not to provide a forgery proof reporting system. It is rather intended as a tool that provides additional measurements, which enable a more realistic test-bed setup.

The WMIReporter uses the *Windows Management Instrumentation (WMI)* to get information about the installed security products from the operating system. WMI provides a uniform interface for managing Windows systems. In WMI, resources, such as the event log, registry, or the file system are organised in object trees that can be accessed using the *WMI Query Language (WQL)*, which is similar to SQL[27] [LM01, chapter 1].

Using WQL, it is also possible to query information about the *Windows Security Center (WSC)*. The WSC collects information about the security state of a Windows system and warns a user if, for example, no firewall is running or the anti-virus software is out of date. The information about the status of these software products are either gathered by the WSC or are reported through a WMI interface by the software products. For example, if an anti-virus product detects that its virus definition database is out of date, it can inform the Windows Security Center about this issue using WMI [Mic08]. The WMIReporter uses this interface to gather information about the security state of the system. Table 7.2 shows which information is available using this interface. Listing 7.2 shows a simplified code example that demonstrates how WMI can be accessed using C# and WQL queries.

Table 7.2: Excerpt of the information available about installed anti-virus and firewall products using WMI

WMI Key	Value for Anti-virus Product	Value for Firewall Product
instanceGuid	C19476...9F7AE8FE	043803A...A79969B
companyName	Avira GmbH	COMODO
displayName	Avira AntiVir PersonalEdition	COMODO Firewall Pro
productUptoDate	true	n/a
enabled	n/a	true
versionNumber	8.0.1.27	2.3.035

```
List<AntiVirusProduct> antivirusProducts = new List<AntiVirusProduct>();
MOS_Searcher = new ManagementObjectSearcher("root/SecurityCenter", "SELECT *
    FROM AntiVirusProduct");
foreach (ManagementObject mos in MOS_Searcher.Get())
{
 antivirusProducts.Add(new AntiVirusProduct(mos["CompanyName"],
     mos["DisplayName"], mos["ProductUpToDate"] == "True",
     mos["versionNumber"]));
}
```

Listing 7.2: Querying the Security Center using its WMI interface

[27] The *Structured Query Language (SQL)* is the standard language for querying relational databases.

The WMIReporter has a simple command line interface that consists of two commands: *firewall check_any* and *antivirus check_any*. Depending on the command, WMIReporter queries the WMI interface of the Windows Security Center via WQL and searches for an enabled firewall or an up-to-date anti-virus product. The result of this process is reported to the OSC-IMC, which passes the result back to the correspondent IMV.

By using WMIReporter as an external measurement tool that is invoked by libTNC's OSC-IMV, more realistic scenarios can be created in a test bed approach. Not only does this allow more complex policies, it also allows changing the security state of a client quickly, for example by disabling the firewall or anti-virus software. Changing the security state of a client effects the decision a VSP is making as part of a TNC integrity check. The next section outlines how the decision making process of the VSP has been implemented.

7.3 Verification Service Provider

Similar to the wTNC client, the VSP has been implemented for this project using C#. To distinguish the general concept of a VSP from its actual implementation, the prototype that has been implemented and which is described in the following is called *wTNC VSP*. Unlike the wTNC client, the wTNC VSP does not have a user interface. As depicted in figure 7.9, it is instead implemented as a command line application. For debugging purposes, the application displays TNCCS-Batches that are exchanged during a TNC check.

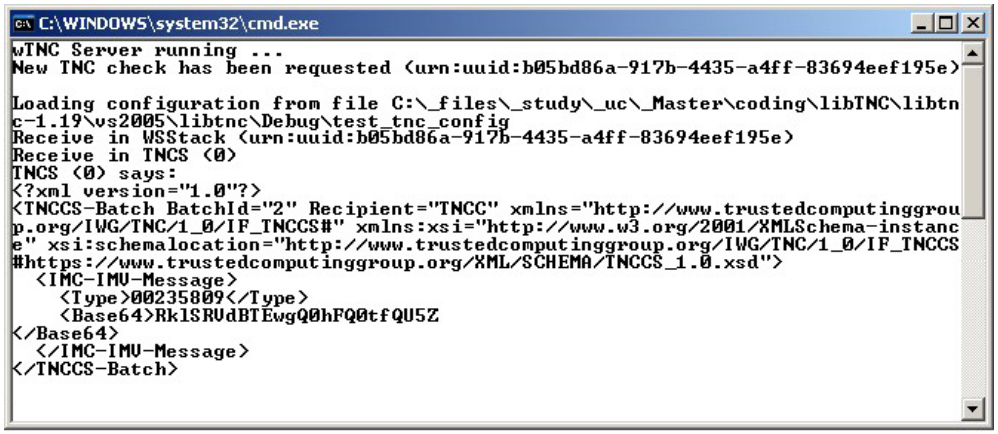


Figure 7.9: Screenshot showing the VSP implementation (wTNC VSP)

7.3.1 Sessions Handling and Encapsulation of libTNC

The mechanism that has been designed and implemented for this project to encapsulate libTNC in the wTNC VSP is similar to the mechanism used by the wTNC client, as described in section 7.2.2.2. However, unlike the wTNC client, the wTNC VSP needs to sup-

port several concurrent TNC connections. This requires that the state of each TNC connection is handled independently. TNCCS-Batches do not support sessions. That is, on the TNCCS-Batch layer, it is not possible to decide whether two given TNCCS-Batches originate from the same client. To overcome this limitation, it is necessary to introduce sessions on the underlying layer, that is, at the communication layer. The communication between VSP and client is based on SOAP. The standard for session management in SOAP messages is *WS-ReliableMessaging (WS-RM)* [Dav08]. WS-RM, which is supported by WCF, allows adding a session ID to a SOAP-based message exchange which allows identifying messages that are sent from the same origin.

Two additional SOAP message are added by WS-RM at the beginning of a message exchange in order to setup a session between two communicating parties. The first message, shown in listing 7.3[28], is sent from a wTNC client to a wTNC VSP. It contains a session ID that the wTNC client expects the wTNC VSP to use in all following messages. The VSP responds to this message by acknowledging the session ID proposed by the client (cf. listing 7.4). It further states which session ID it expects in all further messages that the client sends to the VSP. After the session is established, both wTNC VSP and client include a session ID in every message. The session ID is transferred in an element of the SOAP header, as listing 7.5 shows. This element contains the session ID (*Identifier*) as well as a number which indicates the position of the current message in the message exchange. By including the session ID in every message, the VSP can associate an incoming TNCCS-Batch with a particular client.

```
<CreateSequence> ...
 <Offer>
  <Identifier>urn:uuid:934ed...82b717</Identifier>
 </Offer>
</CreateSequence>
```

Listing 7.3: Session initiation in WS-RM

```
<CreateSequenceResponse>
 <Identifier>urn:uuid:a57b3...c157dd</Identifier>
 <Accept> ... </Accept>
</CreateSequenceResponse>
```

Listing 7.4: Session acknowledgement in WS-RM

```
<r:Sequence s:mustUnderstand="1">
 <r:Identifier>urn:uuid:934ed...82b717</r:Identifier>
 <r:MessageNumber>1</r:MessageNumber>
</r:Sequence>
```

Listing 7.5: WS-RM session ID as present in a SOAP header

[28] See section C for a complete message exchange.

The session ID of an incoming message can be retrieved using a WCF API call. As libTNC is not aware of WCF (or any other mechanism related to transporting TNCCS-Batches), it is necessary to map the WS-RM session ID to a mechanism that is compatible with libTNC. Figure 7.10 shows a UML class diagram that clarifies how this mapping was realised in this project for the VSP. As table 7.3 suggests, the same layered model that has been designed for the wTNC client has been used, thereby abstracting the TNC message exchange from its underlying communication.

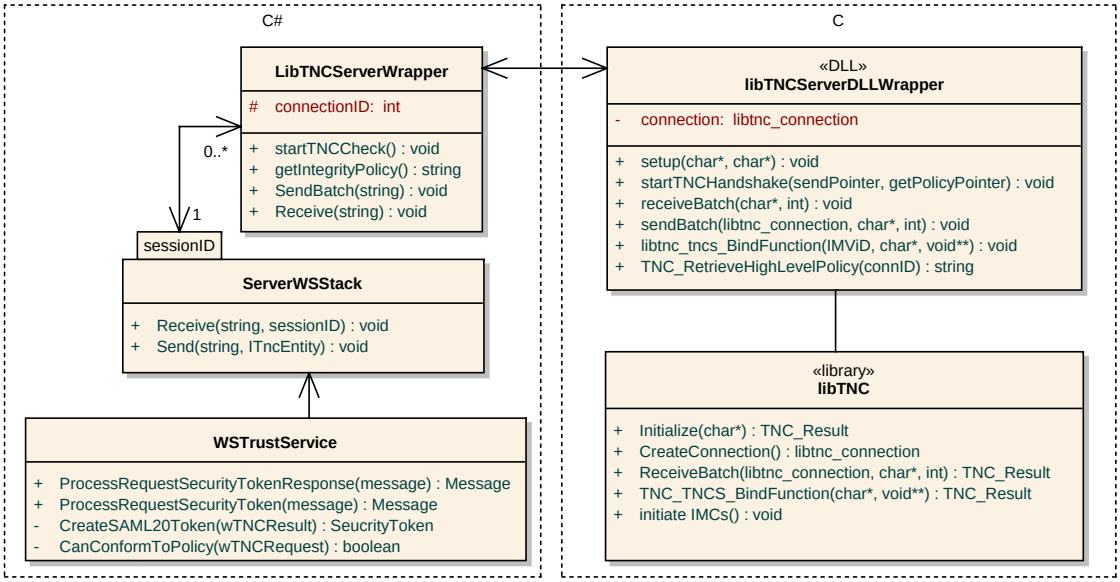


Figure 7.10: Simplified class diagram showing encapsulation of libTNC and session IDs

Table 7.3: Overview of layers and their responsibility in the wTNC VSP architecture

Layer	**Component(s)**	**Responsibility**
TNC Implementation Layer (TIL)	libTNC	Provides an interface to IMCs and the TNC client.
TNC Implementation Encapsulation Layer (TIEL)	libTNCServerDLLWrapper	Provides a TNC implementation dependent interface.
TNC Implementation Abstraction Layer (TIAL)	LibTNCServerWrapper	Provides a TNC implementation independent interface for performing a TNC check.
Communication Abstraction Layer (CAL)	ServerWSStack	Abstracts the underlying communication methods.
Communication Implementation Layer (CIL)	WSTrustService and WCF	Provides support for sending and receiving SOAP messages.

In contrast to the wTNC client, the *Communication Abstraction Layer* (*ServerWSStack*) instantiates a new object in the *TNC Implementation Abstraction Layer* (*LibTNCServerWrapper*) for every new session that is initiated. The newly created object is then associated with the correlating session ID in a mapping table. This allows forwarding an incoming TNCCS-Batch to its *LibTNCServerWrapper* object.

As described in section 7.2.2.2, libTNC has its own connection management. If a new TNC check is requested, a handle in form of a connection ID is returned by libTNC. This connection ID is stored in the corresponding *LibTNCServerWrapper* object. Every time a TNCCS-Batch is delivered to libTNC, this connection ID is supplied in order to associate a TNCCS-Batch with the correlating TNC check.

The communication in the other direction is performed in a similar way as in the wTNC client. On creation of a new TNC connection, libTNC receives a pointer to the corresponding *LibTNCServerWrapper* object. LibTNC uses this pointer to invoke the *sendBatch* method of the corresponding *LibTNCServerWrapper* object.

In the next step, the session ID that belongs to connection ID is retrieved by the *ServerWSStack* and the TNCCS-Batch is sent back to the correct client. Figure 7.11 summarises the relation of session ID and connection ID.

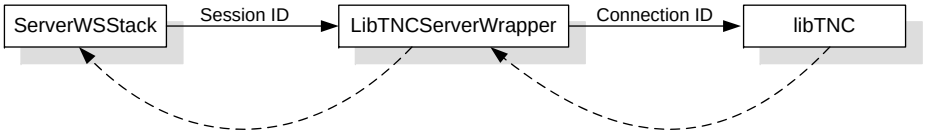


Figure 7.11: Mapping between WS-RM Session ID and libTNC Connection ID

The mechanisms that have been designed and implemented and which have been described in this section enable concurrent TNC integrity checks with libTNC. In order to provide an access recommendation, libTNC requires an integrity policy. The next section describes how this policy is handled in the VSP.

7.3.2 Integrity Policy Handling

In TNC scenarios described in the TNC specification [Tru07b] a TNC server is part of a Policy Decision Point (PDP)[29]. This PDP is equipped with a fixed policy that describes the requirements for users to access a network. In a web-based TNC scenario, however, policies must be handled differently. The TNC server is part of a VSP. This VSP is shared by several parties (SPs), each having different integrity requirements. Instead of handling one fixed policy, a VSP must therefore handle several different policies in a web-based TNC check scenario[30]. According to the TNC specification [Tru07d], the comparison of integrity state and integrity policy requirements is performed by IMVs. Therefore, instead of having one global policy, IMVs must be capable of having one policy per TNC connection.

An integrity policy is stated by an SP in a high level form, as described in section 6.3.3.2. Before this high level policy can be used for the TNC check, it needs to be translated into

[29]Cf. section 2.2.
[30]Figures 3.1 and 3.2 in section 3.1 summarise this.

a low level policy that can be understood by an IMV. This section describes how session-based policy management and policy translation have been implemented in this project.

As described earlier, libTNC provides an example IMC/IMV pair (OSC-IMC and OSC-IMV) that is used as part of the wTNC client and wTNC VSP. The OSC-IMV, which is used within the VSP, has been extended so that it is capable of storing a policy for each TNC connection. The original OSC-IMV loads a policy file when it is initialised. This policy is then parsed and stored in a global variable. The first step for enabling a session-based policy is to change the behaviour of the OSC-IMV. Instead of storing one global policy, the OSC-IMV was modified so that it stores one policy per TNC connection.

Before the OSC-IMV can use an integrity policy received from an SP, it needs to be translated into the IMV specific policy language. All entities shown in figure 7.10 are "candidates" for translating the policy. However, the classes *WSTrustService* and the *ServerWSStack* are not aware of the TNC stack. Implementing the policy translation in these classes would bind the TNC layer to the communication layer, which has been deliberately avoided as discussed above. While the *LibTNCServerWrapper* and the *libTNCServerDLLWrapper* are aware of libTNC, they do not communicate with the IMV layer. Only libTNC communicates with IMVs directly. However, implementing the policy translation within libTNC would bind an IMV to libTNC, which is not intended by the TNC specification [Tru07b]. Furthermore, the TNC specification does not define a policy language for IMVs. As a consequence, each IMV is likely to use its own policy language. LibTNC would need to know the internal policy language of every IMV that it supports to be able to translate the high level policy for this IMV, which is impractical to realise. This leaves the IMV layer for performing the translation. Translating the high level policy at this level has several advantages. Firstly, an IMV is aware of its own policy language and its internal capabilities. Implementing the policy translation at this level therefore encapsulates the knowledge about the policy language in one place. Secondly, an IMV remains independent and can be used with other TNC libraries.

The translation between the high level and the low level policy was implemented in this project using a substitution mechanism. Figure 7.12 shows an example for a policy translation. The grey-scale coding is used to indicate the substitutions that are performed. The high level integrity policy on the left hand side states that it is required that all Windows-based systems have an active personal firewall and an up-to-date anti-virus program installed. On the right hand side, the equivalent OSC-IMV policy representation is given. As it can be seen, the OSC-IMV policy uses a non XML-based approach that uses if-clauses to nest statements.

The nesting of the policy format affects the number of TNCCS-Batches that are exchanged between VSP and client. The OSC-IMV processes all policy statements that are on the same nesting depth in one TNCCS-Batch. The example policy in figure 7.12 therefore results in two TNCCS-Batches that are sent from the VSP to the client. The first TNCCS-Batch

requests the operating system name followed by the second TNCCS-Batch that requests firewall and anti-virus software measurements. The client replies to each measurement request with a TNCCS-Batch that contains the measurement, resulting in a total of four messages being exchanged.

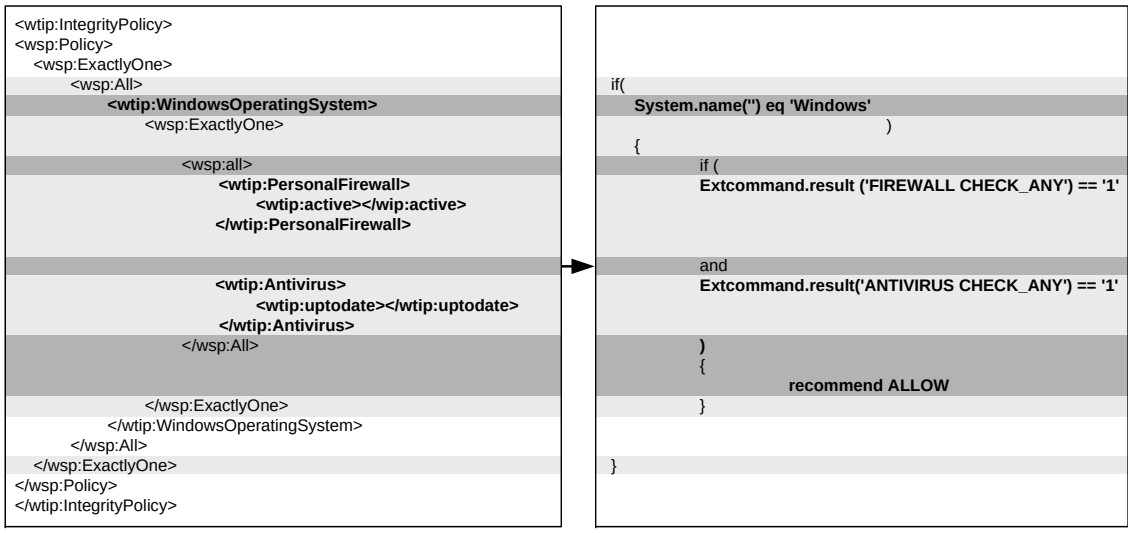


Figure 7.12: Translating the high level policy (left) to the OSC-IMV specific policy language (right)

To avoid that the same policy file is translated several times, a simple caching mechanism has been implemented. This caching mechanism stores the translated policy in a file using the digest (based on SHA1) of the high level policy as the file name. Before a policy is translated, it is checked whether a cached version already exists and, if this is the case, the cached version is loaded from the file system. Otherwise, the policy is translated and its IMV specific representation is saved in a file.

As section 6.3.3.2 describes, the integrity policy language allows the SP to refer to a policy instead of stating it explicitly. Before such a policy can be translated by an IMV, it is necessary to retrieve the policy text. For this purpose, a *PolicyConverter* has been implemented, as depicted in figure 7.13. A policy handler can be programmatically registered with the *PolicyConverter*. If a policy contains a referencing identifier (e.g. a name, an ID, or a URI) for which a policy handler was registered, it will return the referenced policy.

After an integrity policy has been received as part of a TNC check request, it is converted using the mechanism described above, and forwarded to the *LibTNCServerWrapper*[31] object that is responsible for this TNC connection. A callback mechanism, which is similar to the callback mechanism that enables libTNC to send TNCCS-Batches[32], is used to enable the OSC-IMV to retrieve the integrity policy. It is based on an extension mechanism defined in

[31] Cf. figure 7.10.
[32] Described in sections 7.2.2.2 and 7.3.1.

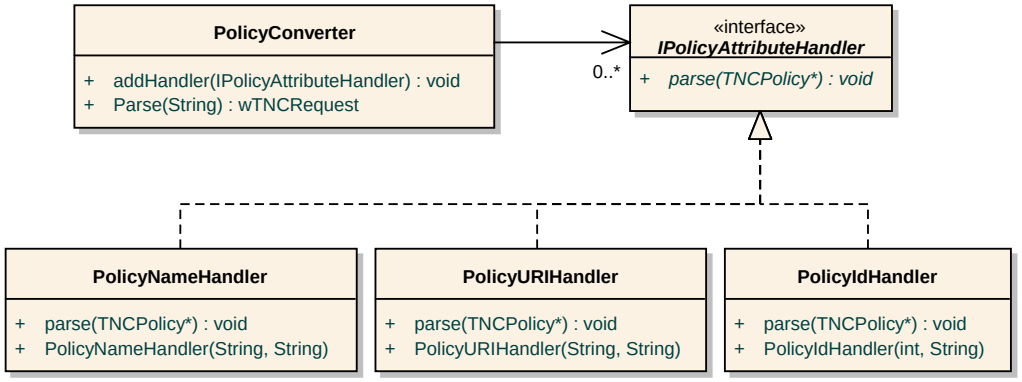


Figure 7.13: Class diagram showing PolicyConverter and registered policy handlers

the TNC specification [Tru07d, section 3.2] and allows the OSC-IMV to dynamically, that is, during runtime, discover the address of a function that provides the integrity policy.

This section summarised the handling of the integrity policy. This policy plays a central role during a TNC integrity check. An overview of how a VSP performs a TNC integrity check is given in the next section.

7.3.3 Performing a TNC check

In the previous sections, the implementation of policy and session handling in the wTNC VSP has been described. This section describes the tasks performed by the implemented wTNC VSP in a chronological order. Figure 7.14 depicts a UML sequence diagram that shows the initiation phase of a TNC check. The wTNC VSP is listening for incoming TNC check requests. As described in section 7.2.2.4, the wTNC client sends the TNC check request encapsulated in a WS-Trust RST message (1.0). On arrival of this message, its content is extracted as described in section 7.2.2.3. Before the TNC check request is processed, the wTNC VSP checks whether it can fulfil the requirements that the SP stated in its VSP policy (message 1.1). If it cannot satisfy the requirements, for example because it is not in possession of a certificate from the requested issuer, the VSP returns a SOAP fault message to the wTNC client. This SOAP fault message contains details about the occurred error and causes the wTNC Client to abort the TNC check with this VSP. If the requirements can be fulfilled, the integrity policy is parsed by the *PolicyConverter* (message 1.2 and 1.3) and the *WSTrustService* object initiates the TNC check. Using its session ID, an incoming TNCCS-Batch is sent to its assigned *libTNCServerWrapper* object (messages 1.4 and 1.5), that was created by the ServerWSStack. Before the TNCCS-Batch can be delivered to libTNC, a new TNC connection is initialised. The messages sent during this initialisation (messages 1.6 - 1.18) are enclosed by a dotted square in the diagram.

As described in the previous section, part of this IMV initiating process is to retrieve the

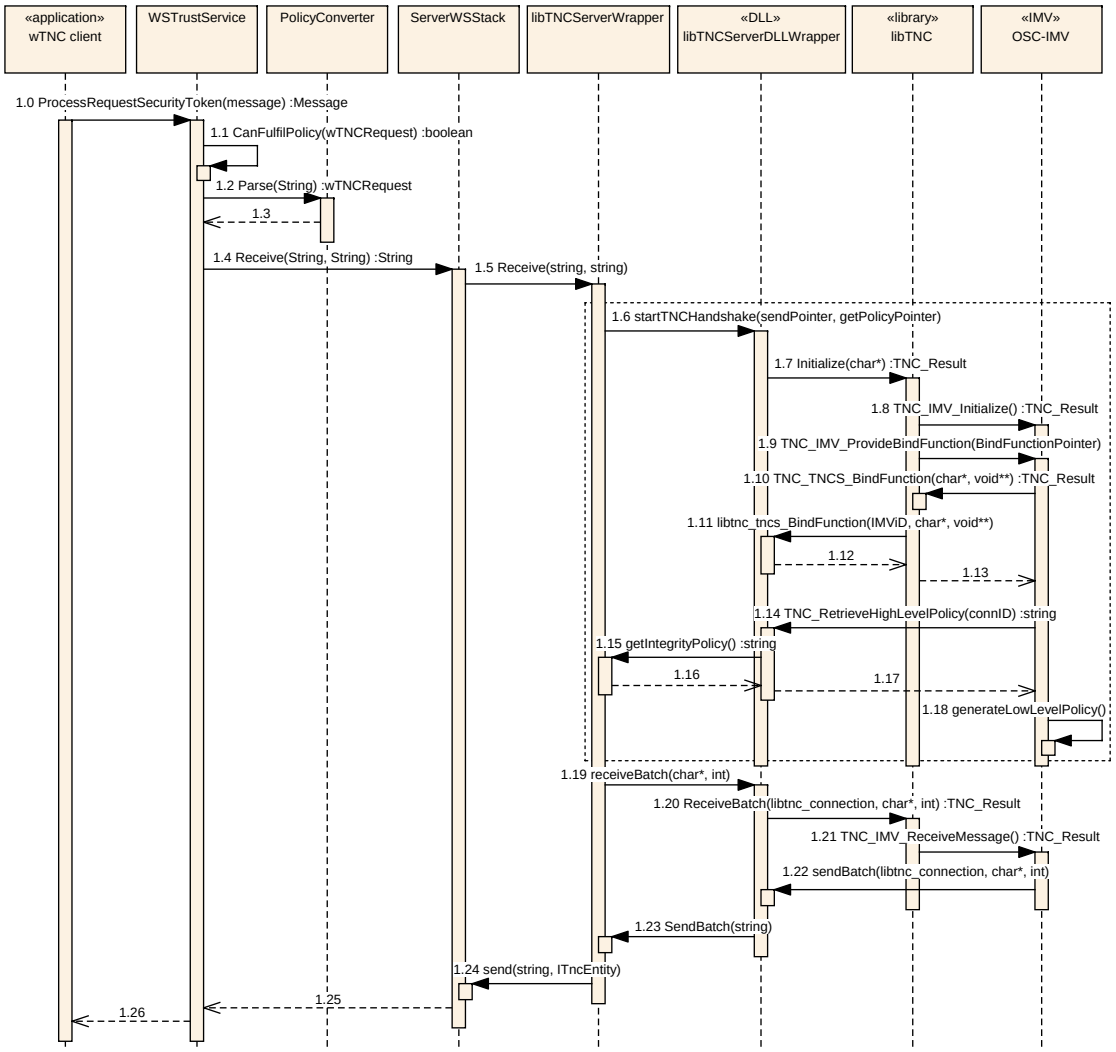


Figure 7.14: Simplified sequence diagram showing initiation of TNC check

high level integrity policy. Before an IMV can retrieve the policy, it needs to retrieve the address of the method that provides this policy. The TNC specification describes an extension mechanism to achieve this. This mechanism, which has been implemented for this project, provides an IMV with a pointer to a lookup method (message 1.9). The IMV calls the provided method (message 1.10) and requests a pointer for the *TNC_RetrieveHighLevelPolicy* method that is part of the *libTNCServerDLLWrapper*. Because libTNC is not aware of this method, it forwards the request to the address lookup method provided by the *libTNCServerWrapper* layer (message 1.11). The requested address is returned (message 1.12 and 1.13) and the integrity policy is retrieved in messages 1.14 – 1.17 and translated into the IMV specific format (message 1.18).

The further message exchange is similar to that of the wTNC client. A TNCCS-Batch is delivered to libTNC via *receive* calls (message 1.19 – 1.21). The TNCCS-Batch that contains

the reply to the incoming message is returned to the wTNC client (message 1.22 – 1.26) using the mechanism described in section 7.3.1.

TNCCS-Batches are exchanged between wTNC VSP and wTNC client until the VSP can make an access decision. The process of issuing this decision is described in the next section.

7.3.4 Issuing Recommendations and SAML Assertions

The previous sections gave an overview of steps that are performed during a TNC check. The last step in a TNC check, that is, the encapsulation of the result, is covered in this section. An overview of this process is depicted in figure 7.15 in a simplified UML sequence diagram. The wTNC VSP receives a final measurement from a client in message 1.0. These measurements are sent to and processed at the TNC layer, as described in the previous section (message 1.1 and 1.2).

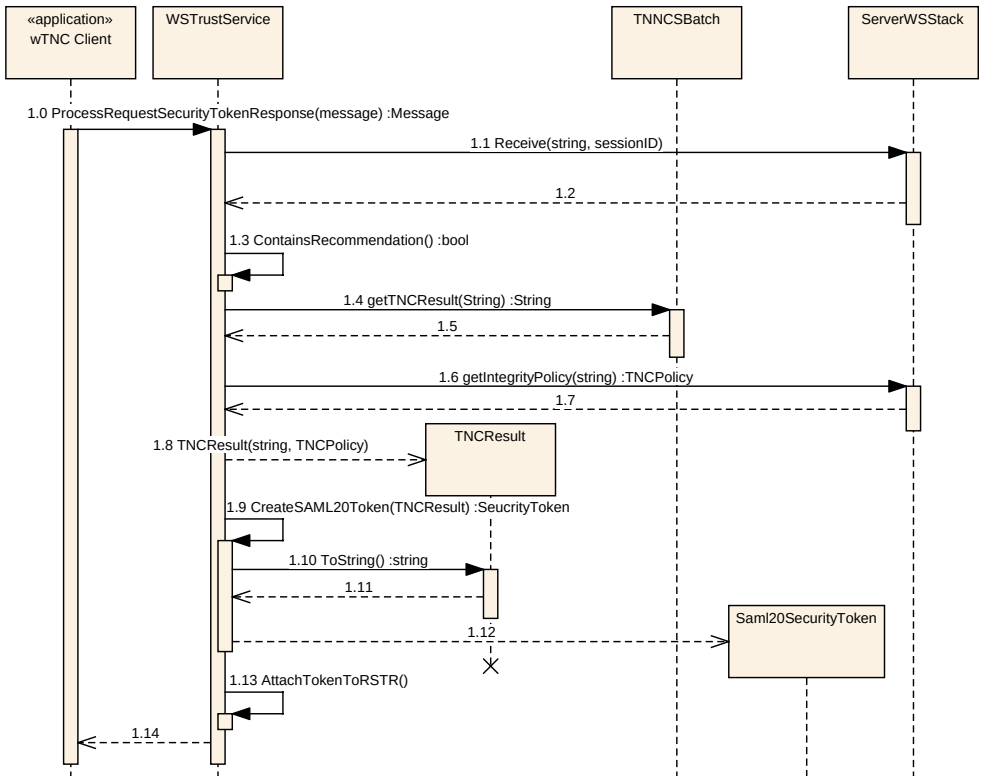


Figure 7.15: Sending TNCCS-Recommendation and SAML token

The *WSTrustService* checks whether a TNCCS-Batch contains a *TNCCS-Recommendation* statement (message 1.3) before forwarding it to the wTNC client. This statement marks the end of a TNC check and it is contained in the last message that will be sent to a client in this TNC check. As described in section 4.3.3, this message contains the TNC check result, which is encapsulated in a SAML assertion that is attached to this message.

The format for the TNC result has been proposed and described in section 4.3.3. It contains two elements, that is, the TNC check result and a representation of the integrity policy that was used to derive the TNC check result. Using the class *TNCCSBatch*, a string representation of the TNC check result is created (messages 1.4 and 1.5). In addition, the integrity policy is retrieved, which is stored in the *ServerWSStack* for the duration of a TNC check (messages 1.6 and 1.7).

The TNC check result is represented as an object of the class *TNCResult*. This object is instantiated using the TNC result string and the integrity policy (message 1.8). The XML representation of this object, that is, the TNC check result for the SP, is encapsulated using a SAML assertion. As described in section 4.3.3, the SAML assertion uses the *bearer* confirmation method. The SAML implementation that is part of WCF, however, does not support this method. It was therefore necessary to use an external SAML library. An existing SAML library has been found that supports the bearer confirmation method[33]. Furthermore, this SAML library supports the SAML 2.0 specification. Using the API of this library, the *WSTrustService* creates the SAML assertion in the method *CreateSAML20Token* (message 1.9). Listing 7.6 shows an excerpt of this method. In this excerpt, the validity period of the SAML token is set to 10 minutes (lines 3–6). Furthermore, the TNC result is included as an attribute statement (line 8-12).

The SAML library does not support "raw", that is, non-escaped XML data in attribute statements[34]. This functionality was added to the library for this project. Line 11 in the code excerpt shows how this extension can be used.

The TNCResult object is serialised into its XML representation using the implemented *toString* method in line 11 (message 1.10 in figure 7.15). This method produces an XML fragment in the format described in section 4.3.3.

A signature is added to the token using the key included in the certificate which was requested by the SP. The SAML assertion is now complete and can be attached to the RSTR message that is sent to the wTNC client (message 1.13). The client extracts the SAML assertion and forwards it to the SP. For this project a sample SP has been implemented, as described in the following section.

[33]This library, similar to the WS-Trust implementation that is used in this project, is part of a sample implementation provided by Microsoft and is available from http://wcf.netfx3.com/files/folders/authorization/entry11435.aspx.

[34]XML has reserved characters, such as "<" and ">", which are used to enclose an element's name. Each reserved character has an associated escape sequence. By using the escape sequence of a reserved character (for example, "&lth;" for the "<" character), the character's literal representation is encoded. However, by escaping all reserved characters, the semantic of an XML document or fragment is changed during the process. This prevents, for example, that an XML document is validated and compared to its XML Schema representation.

```
1  Saml20Assertion samlAssertion = new Saml20Assertion();
2
3  Saml20Conditions samlConditions = new Saml20Conditions();
4  samlConditions.NotBefore = DateTime.UtcNow;
5  samlConditions.NotOnOrAfter = DateTime.UtcNow.AddMinutes((double)10);
6  samlConditions.Conditions.Add(condition);
7
8  Saml20AttributeStatement samlAttributeStatement = new Saml20AttributeStatement();
9  Saml20Attribute tNCResultAttribute = new Saml20Attribute();
10 tNCResultAttribute.Name = Constants.TNCResult.Name; \\"TNCResult"
11 tNCResultAttribute.AttributeRawValues.Add(tncResult.ToString());
12 samlAttributeStatement.Attributes.Add(tNCResultAttribute);
13
14 samlAssertion.Conditions = samlConditions;
15 samlAssertion.Statements.Add(samlAttributeStatement);
```

Listing 7.6: Code excerpt showing aspects of the creation of a SAML assertion

7.4 Service Provider

In the web-based TNC model, an SP offers a service using a Web interface. This service can be accessed using a standard Web browser. In order to test the interaction of a client with a service provider, a Web application has been developed. This implementation of the fictitious Web site *ACME secure Web* provides support for requesting a TNC integrity check and analysing the resulting security token. It is an example implementation of an SP that is substitutional for an online banking service, a company's web-based Intranet, or any other security sensitive Web application that makes use of TNC. The Web application has been developed using ASP.Net. The Web pages that are generated by the SP can be accessed using HTTPS and are hosted on an Internet Information Server (IIS).

In the following, an overview of how the SP has been implemented is given. Section 7.4.1 shows the SP from a user's perspective, while section 7.4.2 describes how the security token that contains the TNC check result is validated.

7.4.1 User Experience

The user interacts with the SP using a standard Web browser. Figure 7.16 shows an overview of the Web pages that have been developed for the SP. Before a user can use the service offered by an SP (that is, the *Service* Web page), an authentication has to be performed. Figure 7.17 shows a screenshot of the login screen, where a user logs in using his or her user name and password. After being successfully authenticated by the SP, the user is transferred to a Web page that prompts the user to perform a TNC check (depicted in figure 7.18). After clicking the *Start TNC check* button, the wTNC clients is invoked by the wTNC browser extension and the TNC check can be performed as described in section 7.2.2. On successful completion of the TNC check, its outcome is presented to the user, as shown in

the screenshot in figure 7.19. Depending on the outcome, the user is granted or denied access to the service. It is worth pointing out that this scenario demonstrates only one of several possible implementation choices that an SP has. For example, authentication and TNC check request could be performed in one step. Furthermore, instead of denying access to a service if a client does not comply to the integrity policy, it would be possible to only partially allow access. These are implementation choices that can be made by each SP independently.

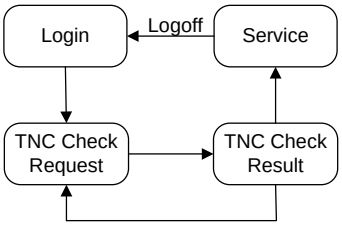


Figure 7.16: Site map showing the Web pages of the ACME secure Web SP implementation

In addition to offering a service, an SP must state its policies and validate the security token that it receives from the VSP through the client. The process and syntax of stating policies have been described in section 6.3.4. The next section covers the steps that are undertaken to validate a token and retrieve the TNC check result.

7.4.2 Token Validation

After the client has performed a TNC check with a VSP, it sends the check result encapsulated in a security token to the SP, as described in section 6.3.4. Before an SP interprets the TNC check result and concludes about the client's integrity state, it must be ensured that the token is valid. This validation check consists of three main steps.

1. Validating the cryptographic signature of the token.
2. Verifying that the issuer of the token is trusted.
3. Validating that the TNC check was performed according to the correct integrity policy.

The implementation of these steps is summarised in figure 7.20. This figure shows a simplified UML sequence diagram that outlines the creation of the Web page, which indicates the TNC check result to the user (named *TNCCheckResult.aspx* and depicted in figure 7.19). This Web page (*TNCCheckResult.aspx*) is requested by the user after the TNC check has been performed. Along with the HTTP request for this page, the browser sends the security token as a HTTP POST parameter[35]. The Web server forwards the request and its parameters

[35]Cf. sections 5.3.1 and 6.3.4.

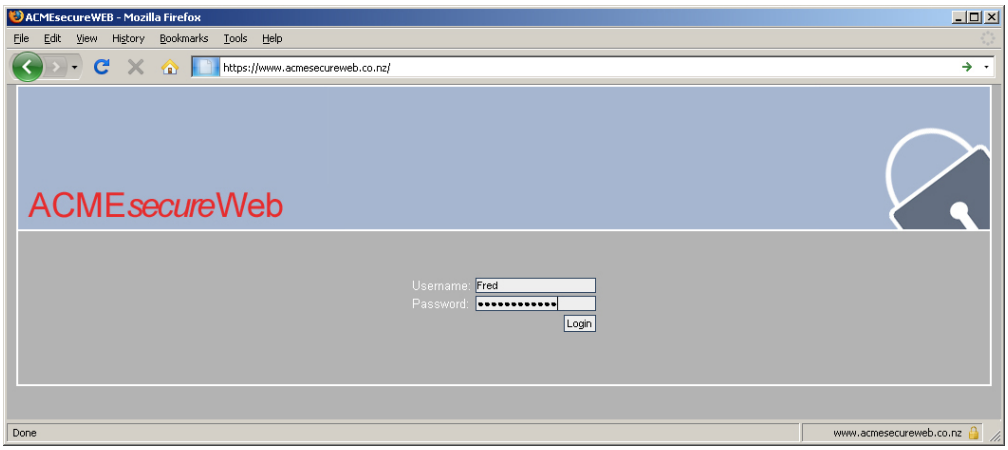


Figure 7.17: Screenshot showing the login screen generated by the SP

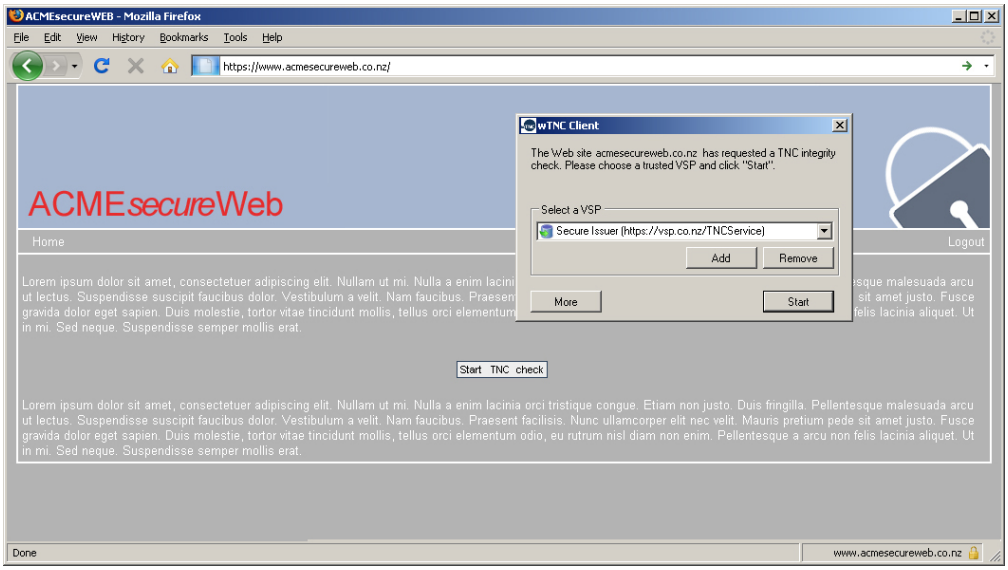


Figure 7.18: Screenshot showing the Web page that triggers a TNC integrity check

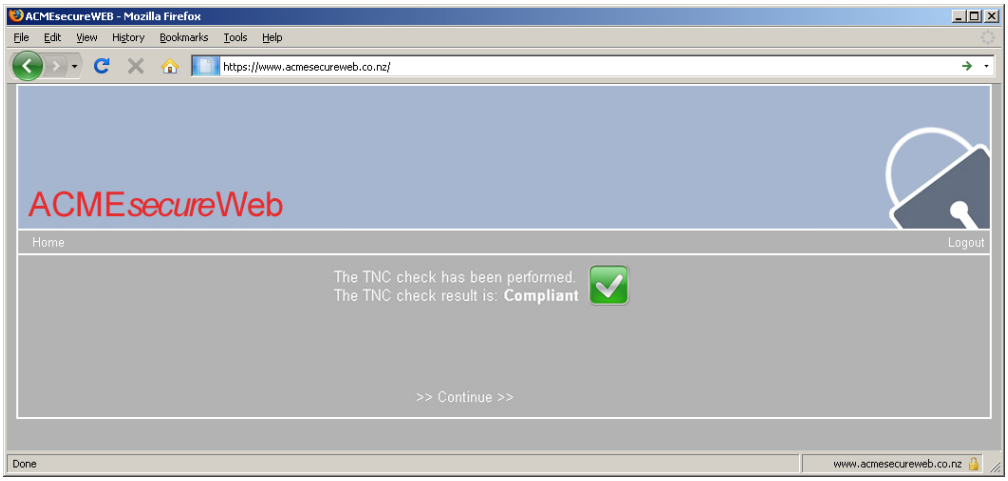


Figure 7.19: Screenshot showing the result of a TNC integrity check

to the .Net framework, which will invoke the *Page_Load* method of the *TNCCheckResult* class (message 1.0 in figure 7.20). This class is responsible for handling requests and responses for the *TNCCheckResult.aspx* Web page.

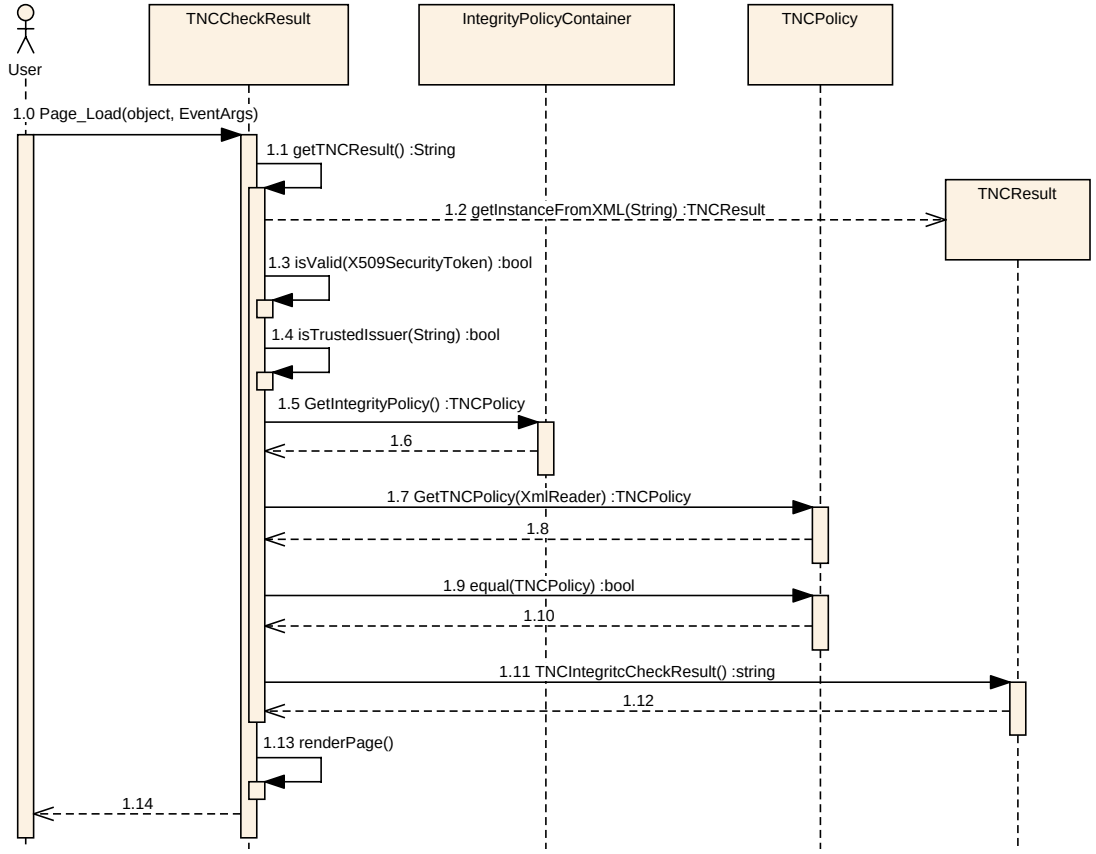


Figure 7.20: Sequence diagram showing the process of validating a security token

The first step in retrieving the TNC check result (message 1.1) is to parse the security token and convert it into a *SecurityToken* object. As part of this process the validity period of the security token is checked. The *SecurityToken* data structure allows verifying the signature that is attached to the token. The process of verifying such an XML signature has been outlined in section 4.3.2.2. If the signature matches the token, the next step is to check which entity has issued the token and whether this entity is trusted by the SP (message 1.3).

The identity of the issuer can be determined using its X.509 certificate. As described in section 4.3.2.2, an XML signature includes an element that points to the certificate which contains the key that was used to create the signature. Consequently, the SP needs to locate this certificate in order to validate the signature. Two possibilities exist for locating a certificate. If the XML signature contains only a reference to the certificate (e.g. a name or a thumbprint) then the SP must have a local copy of the certificate in its certificate store. If such a local copy does not exist, the VSP needs to include the certificate into the XML signature. The SP can indicate whether it needs a certificate to be included using its VSP

policy[36]. In either case, the SP can access the certificate and extract the containing public key for validating the signature.

For this prototype implementation, the process of certificate validation was simplified. In particular, a mechanism for checking whether a certificate has been revoked was not implemented. Therefore, the process of validating a certificate is reduced to validating its certificate chain. In order to succeed, all certificates in the chain must be present on the SP and stored in the certificate store[37]. If the certificate chain is successfully validated, the certificate that was used to create the signature was issued by a trusted CA and the identity of the VSP is verified.

In the next step, the information retrieved through the certificate is checked against the VSP policy, that is, it is checked whether the VSP that issued the security token is trusted for performing the TNC check. If an explicit policy[38] is used, it is checked whether the VSP is listed as one of the trusted VSPs. For implicit policies, it is checked whether the issuer of the certificate is contained in the list of CAs that are trusted for issuing certificates to VSPs (message 1.4). In contrast to the certificate store that contains trusted CAs, this list is maintained within the SP application. If a match between VSP identity (or their CA in the implicit case) and the VSP policy can be found, it is verified that the TNC check was performed by a VSP that is trusted for this purpose.

At this stage in the validatation process, the SP has verified that the TNC result was not altered (for example by the client) and that a trusted VSP has performed the TNC check. The last step before the TNC check can be trusted is to verify that the integrity policy that was sent to the VSP was not altered. For this purpose, the current integrity policy is retrieved from the class *IntegrityPolicyContainer* (message 1.6) and compared to the policy representation that is contained in the TNC result (messages 1.7 and 1.8). As described in section 4.3.3, instead of returning the entire integrity policy, only its hash value is returned to the SP. The SP must therefore create the hash value of its integrity policy and compare it to the hash value contained in the TNC check result. If these values match, the TNC check result was validated successfully and can be trusted. It is retrieved from its data structure (message 1.11) and the Web page which is depicted in figure 7.19 is generated (message 1.13) and returned to the user (message 1.14). The outcome of the TNC check is stored in the user's session data. When the user accesses other Web pages from the same SP, the session data is processed and the user is granted access to pages without requiring to re-perform a TNC integrity check.

[36] Cf. section 6.3.3.3.
[37] Further information about where to store these certificates is given in the installation instructions for the SP in section A.2.
[38] Cf. section 4.4.

7.5 Test Bed and Evaluation of the Prototype Implementation

The component described in the previous sections have been implemented and combined in a test bed in order to evaluate their behaviour. This test bed is depicted in figure 7.21. In addition to functional tests, performance tests have been undertaken to evaluate various aspects of a web-based TNC check. An important aspect, discussed in section 7.5.1, is the total time that it takes for a user to successfully perform a web-based TNC check. During a TNC check, each entity (VSP, SP, and client) performs various tasks. Section 7.5.2 shows how much time each party spends on the performance of each of its tasks. Finally, section 7.5.3 shows the results of load tests that have been performed to evaluate the scalability of the implemented prototype.

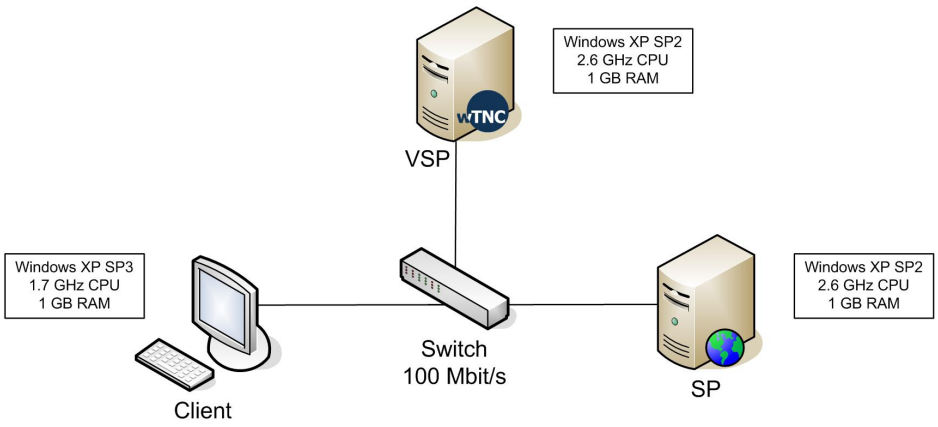


Figure 7.21: Overview of the test bed and its components

7.5.1 End-to-End Time

This section shows the end-to-end time of a web-based TNC check, that is, the total time that elapses from initiating the TNC check through the browser until the browser displays the SP's access decision.

From a user's perspective, the TNC check can be divided into three phases.

Initialisation This phase starts with initiating the web-based TNC check using a browser. The wTNC client is triggered and the user must select a VSP. The phase finishes when the user starts the TNC check[39].

TNC Check In this phase, integrity measurements are sent from the client to the VSP. Furthermore, the VSP sends the TNC check result to the client. Finally, the wTNC client is terminated and the browser is notified.

[39]Cf. figure 7.4 on page 111.

Result After the wTNC client is terminated, the browser submits the TNC check result encapsulated in a security token to the SP. The SP assesses and validates the token before making an access decision. After the decision has been made, the SP generates a Web site that informs the user about the outcome of the TNC check. The Web site is retrieved and displayed by the user's browser.

In order to measure the duration of each phase, it is necessary to collect performance measurements from the user's perspective, that is, through the browser. However, a browser cannot be used to store this information persistently, for example, in text files. In order to circumvent this restriction, a modified version of the wTNC browser extension[40] has been created and used in this test. The wTNC browser extension was extended so that it can communicate with Javascript that is part of a Web site. When the user clicks the button that initiates the TNC check, a Javascript event is triggered. This event is intercepted by the modified browser extension which stores the event and the current time (in milliseconds) into a file. Furthermore, the wTNC client was modified for this test. It records the current time when a user starts the TNC check and when it receives the TNC check result from the VSP. In order to automate the test, further modifications have been made. The wTNC client starts the TNC check automatically using a preselected VSP. The Web pages that are produced by the SP have been modified, so that the TNC check starts automatically without user interaction. Furthermore, the Web page that states the TNC check result redirects the Web browser to the Web site that initiates the TNC check. These modifications allowed gathering measurements automatically, that is, without user interaction, which improved the repeatability of the experiment.

The end-to-end time partially depends on how many different measurements are performed during a web-based TNC check. In order to determine the influence of this factor, the tests have been performed using six test cases, as summarised in table 7.4.

Table 7.4: Measurements performed during each test case and number of resulting TNCCS-Batches

Case	Performed Measurement	Measurement Method	TNCCS-Batches
1	Operating system version	OS-layer call	2
2	OS version and file information	OS-layer calls	4
3	OS version and file information (2)	OS-layer calls	6
4	Virus scanner status	WMIReporter	2
5	Virus scanner and firewall status	WMIReporter	4
6	Virus scanner and firewall status (2)	WMIReporter	6

In each test case different measurements have been performed that affect the number of TNCCS-Batches that need to be exchanged during a TNC check. In addition to the number of messages, also the type of measurements varies in the test cases. In the first three test cases the IMC collects status information by calling operating system functions. These functions are used to obtain the current version of the operating system and the status of

[40] Cf. section 7.2.1.

files in the file system. In the last three test cases the external application *WMIReporter*, as described in section 7.2.3, is used to gather information about the status of virus scanner and firewall products.

In this experiment a single user accesses the SP and VSP, that is, no additional load is generated. Test data was collected in 550 test runs for each test case. In order to limit the effects of caching and thread ramp-up time[41], the results of the first 50 test runs were not taken into consideration. The mean times recorded in this experiment are specific to the hard- and software combination used in the test bed. Nevertheless, the results give an indication about how long it takes a user to complete a TNC check. Figure 7.22 shows an overview of the results of this test. Table 7.5 contains more details about the measured times in each phase.

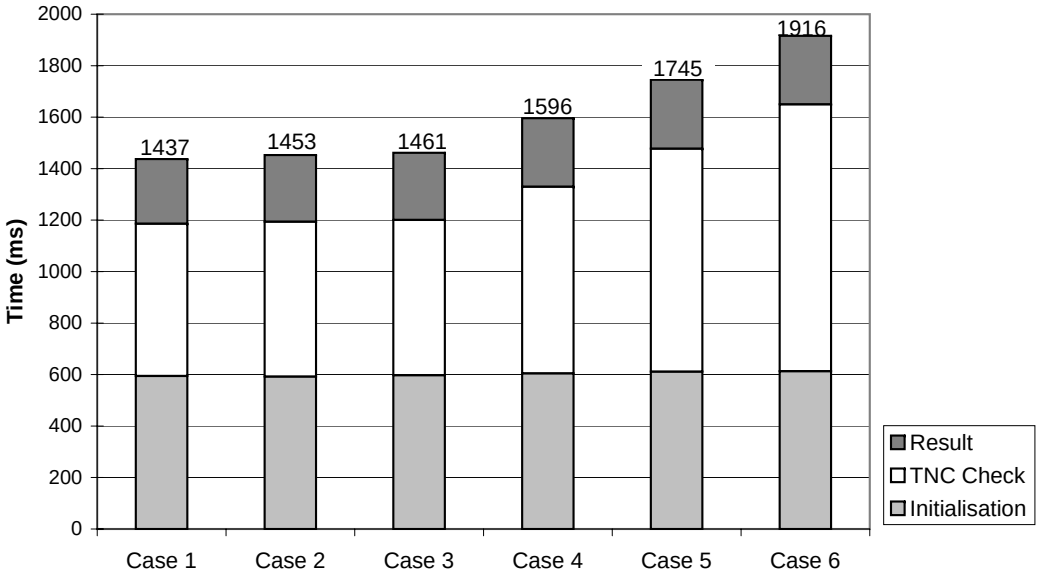


Figure 7.22: Results of the end-to-end time performance test

Table 7.5: Mean times (in ms) measured during the end-to-end test

Case	Initialisation (ms)	TNC Check (ms)	Result (ms)	Total (ms)
1	594.4	591.9	250.8	1437
2	592.2	601.9	259.0	1453
3	597.4	603.3	260.7	1461
4	605.0	724.7	266.3	1596
5	611.5	866.3	267.3	1745
6	613.3	1036.5	266.3	1916

The recorded times are slightly higher than the actual processing time, because they include the time that it takes to store the performance measurements into a text file. In the first three test cases, the TNC check is completed in less than 1.5 seconds for a single user. As

[41] That is, the time it takes to set up new threads to handle a certain amount of requests.

mentioned above, this times include all steps from initiating the TNC check to receiving the Web page that the SP returns and which contains the access decision. On average, it takes 602.3ms to process the TNC check request, load the wTNC client application, and start the TNC check (initialisation phase). After the TNC result has been retrieved, the result phase is initiated, in which the SP receives and evaluates the security token and generates a Web page that is rendered by the browser. In average, this phase is completed within less than 262ms.

It can be seen that increasing the number of exchanged TNCCS-Batches does not notably affect the processing time for the TNC check phase (24ms (1.9%) from case 1 to case 3). By contrast, increasing the number of measurements and TNCCS-Batches when using the WMIReporter application affects the time for performing the TNC check phase. Increasing the number of TNCCS-Batches from 2 to 4 (case 4 and case 5) when using the WMIReporter increases the total time by 149ms (19.5%). Similarly, increasing the number of TNCCS-Batches from 4 to 6 (case 5 and case 6) increases the total time by 174ms (19.6%).

The mean times for the initialisation and result phases are slightly increased when comparing case 1 and case 6 (18.9ms (3.2%) and 15.5ms (6.2%) respectively). These results are unexpected as these phases are not affected by increasing the number of TNCCS-Batches or measurements. However, the increased mean times for these phases can be explained by a higher CPU load on the client when performing more computationally expensive integrity measurements. This effects the load time for the wTNC client and the time the browser requires to send the security token to the SP and render the returned Web page.

In summary, the obtained measurements suggest that the effects of exchanging more TNCCS-messages are neglectable in a single user scenario. The effects of the WMIReporter, however, are clearly visible in the test results. It is therefore essential to limit the effects that integrity measurement software has on the client as it can otherwise notably delay the TNC check process. Overall, the TNC checks that have been performed in this test were completed in reasonable time. [BKB00] concludes that a user is willing to wait 8 seconds before deciding to abort the process of loading a Web page. In this test, all TNC checks were completed in under 2 seconds. While this time is specific to the test bed and does not include network latencies typical for internet connections, it shows that a web-based TNC check can be performed without notably disrupting a user gaining access to a service.

The test phases in this experiment give a rough indication about how the processing time is distributed during a TNC check. The next section shows the results of a performance analysis that shows the distribution of processing time in each entity.

7.5.2 Component Performance Analyses

Each entity in the attestation-based architecture, that is, the client, the VSP, and the SP, perform several tasks during a web-based TNC check. This section presents the results of a

performance test in which the processing time of key tasks has been analysed. In order to obtain these measurements, a profiler application[42] has been used. This profiler "hooks" into a running application and can protocol the execution time of each line of code. A side-effect caused by the overhead a profiler creates is that the execution time of the examined application drastically increases. During a profiler run, for example, the VSP implementation took about 12 seconds to complete a TNC check, compared to a few hundred milliseconds without a profiler. It is therefore not possible to provide information about the absolute execution time. However, the delays caused by a profiler are affecting each statement in an application equally. This allows an assumption to be made about the execution time of a task in relation to the total execution time. As such, the execution time of a task can be expressed as a percentage of all tasks that have been performed during a test run.

For the following tests, an integrity policy has been used that requires a certain operating system version (retrieved via an OS-API call) and an up-to-date virus scanner (retrieved via WMIReporter). In total, 4 TNCCS-Batches are exchanged between client and VSP. The client is compliant to the policy, which the SP can infer from the issued SAML assertion. In the following, the results for each entity are summarised.

7.5.2.1 wTNC Client

Figure 7.23 shows an overview of the tasks performed by the wTNC client during a TNC check. Table 7.6 contains detailed measurement values of these tasks and their sub tasks.

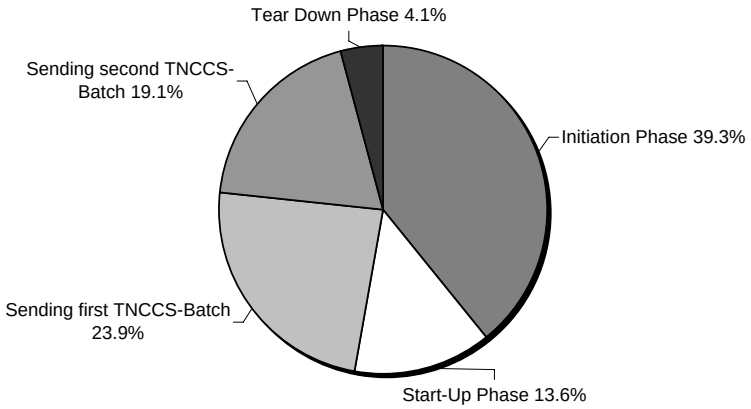


Figure 7.23: Tasks performed by the client and related percental execution time

The majority of time (39.3%) during a TNC check in this test is spent in the initiating phase, that is, while loading the wTNC client application. This phase consists of two main blocks, that is, generating and rendering the GUI (24.7%) and generating the VSP status list (14.6%)[43].

[42] ANTS Profiler - http://www.red-gate.com/products/ants_profiler/index.htm.
[43] Cf. figure 7.5 on page 113.

Table 7.6: Tasks and subtasks performed by the client and related percental execution time

Initiation Phase	**39.3%**
Initialise GUI	24.7%
Parse policies and setup VSP selection	14.6%
Start-Up Phase	**13.6%**
Initiate libTNC	0.3%
Initiate WCF	13.3%
Sending first TNCCS-Batch	**23.9%**
LibTNC processing time	0.4%
Establish Connection (SSL & WS-RM)	17.6%
Create and send WS-Trust RST message	5.9%
Sending second TNCCS-Batch	**19.1%**
LibTNC processing time	2.3%
WMIReporter	7.4%
Create and send WS-Trust RSTR message	0.9%
Decode security token	8.5%
Tear Down Phase	**4.1%**
Store security token	2.2%
Close libTNC	1.9%

The second phase (start-up phase) begins after the user starts the TNC check. During this phase underlying libraries are initiated. While initiating libTNC only requires 0.3% of the total execution time, loading the communication framework WCF requires 13.3%.

Preparing the first TNCCS-Batch requires slightly more time compared with the second TNCCS-Batch. Most of the time (17.6%) in this phase is spent on initiating the SSL connection and establishing the WS-RM session[44]. Performing the first measurement (querying the operating system version) only requires 0.4% of the total time compared to 5.9% for encapsulating the TNCCS-Batch into a WS-Trust RST message.

The second measurement relies on the WMIReporter. Performing the measurement takes 7.4% of the total execution time. Using the WMIReporter also increases the execution time for libTNC, as it needs to launch the WMIReporter application, which introduces additional overhead. The reply from the VSP contains the security token for the SP. In order to validate its format, it is transferred into a data structure. This process requires 8.5% of the total execution time.

Finalising the TNC check, that is, storing the token and closing libTNC requires 4.1% of the total execution time.

It is noticeable that the wTNC client spends most its execution time with loading, setting up libraries, and establishing connections (around 70%). This time could be shortend by preloading parts of the application. Such an approach requires that the wTNC client application or parts thereof continually run in the background and therefore consume memory. If a TNC check is not frequently performed, this approach might therefore not be feasable.

[44]Cf. section 7.3.1.

Furthermore, handling the SAML token on the client requires 10.7% of the total execution time. This is more time than is spent on performing the measurements (10.1%). However, the SAML token proccessing costs are likely to be related to the experimental status of the underlying SAML 2.0 library. Once a suitable SAML library is available in WCF, this time is likely to decrease.

The counterpart to the wTNC client is the VSP implementation. The performance of its phases during a TNC check is discussed next.

7.5.2.2 Verification Service Provider

In this test run the same policies are used as in the previous client test run. Figure 7.24 depicts an overview of the tasks and their relative duration that are performed during a TNC check. Details about subtasks can be found in table 7.7.

Unlike the client, a VSP server is typically started once and performs many TNC checks. This means that the weight of a one-off initiation phase decreases if more TNC checks are performed. The results in figure 7.24 reflect the results for a scenario in which one TNC check performed.

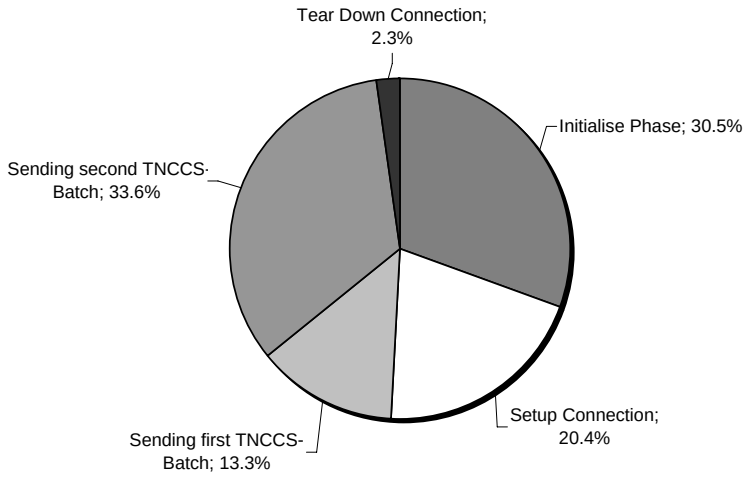


Figure 7.24: Tasks performed by the VSP and related percental execution time

In this scenario, the initialisation phase requires 30.5% of the total execution time. The substask of setting up end loading WCF required the largest proportion of time (19.5%). In constrast to the initialisation phase, the setup connection phase is performed for every TNC check. In this phase the connection with the client is established (17.6%) and libTNC is loaded (2.8%).

After the connection has been established, the client sends the first message. Processing this message requires 13.3% of the total processing time. This time includes parsing and converting the integrity and VSP policy (5.5%), using libTNC to compare the measurement

to the integrity policy (0.4%), and creating the WS-Trust RSTR message for sending further measurement requests to the client (5.5%).

Processing the second TNCCS-Batch requires more processing time (33.6%) than the first TNCCS-Batch. This is largly related to the SAML assertion, which is generated for encapsulating the TNC check result (19.8%). Furthermore, creating the RSTR message requires an additional 11.7%. These processing times are considerable longer than the actual time spent on comparing the measurement with the integrity policy (0.4%).

After a TNC check is finished, related resources and connection need to be released. This process requires 2.3% of the total processing time.

Table 7.7: Tasks and subtasks performed by the VSP and related percental execution time

Initialise Phase	**30.5%**
Setup WCF	19.5%
Preparing Connections	11.0%
Setup Connection	**20.4%**
Creating WCF Channel	17.6%
Setup libTNC	2.8%
Sending first TNCCS-Batch	**13.3%**
Convert Integrity Policy	2.9%
Check conformance to VSP policy	2.6%
Parse incoming RST message	1.9%
libTNC processing time	0.4%
Create RSTR message	5.5%
Sending second TNCCS-Batch	**33.6%**
Parse incoming RSTR message	1.8%
libTNC processing time	0.4%
Create SAML assertion	19.8%
Create RSTR message	11.7%
Tear Down Connection	**2.3%**
Unload TNC Session	0.4%
Close WCF Connection	1.9%

7.5.2.3 Service Provider

As intended by the assertion-based attestation model, the SP has to fulfil the least number of tasks compared to the client and the VSP. Figure 7.25 shows an overview of the tasks and their percentual execution time that are performed while evaluating the SAML token issued by the VSP. The tasks in this figure are fine grained and are not further broken down into subtasks.

The results of this test reveal that the SP spends most of the time (40.8%) parsing the SAML token, that is, converting it into a data structure. Similar to the client and the VSP, it is likely that this relatively high execution time is related to the used library. After the token has been

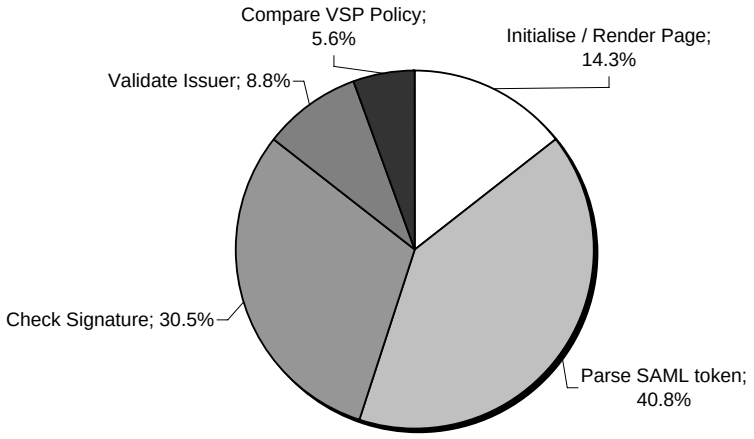


Figure 7.25: Tasks performed by the SP and related percental execution time

parsed, its signature can be validated, as described in section 4.3.2.2. This process requires the second largest proportion of the total execution time (30.5%). This figure reflects the high computanional costs involved in performing cryprographic operations[45].

The remaining tasks (28.7%) are devided as follows. Validating the issuer of the SAML assertion, that is, validating the certificate chain and ensuring that the issuer is trusted for performing a TNC check for the SP requiries 8.8% of the total execution time[46]. Before the assertion from the VSP can be used in the SP's access decision, the SP compares the requested integrity policy with the policy returned from the VSP (5.6%). Finally, generating the Web page that contains the access decisions requires 14.3% of the total processing time.

7.5.3 Load Tests

The previous section gave an overview of the relative execution times of tasks performed during a TNC check. In order to bring these measurements into perspective, an experiment was conducted that focusses on absolute execution times. This section presents the results of this experiment, which also shows how the VSP and the SP perform with increased load and parallel requests.

For generating requests and measuring response times for the SP test case, the open-source tool JMeter[47] has been used. It is able to simulate a configurable number of users by launching simultaneous threads that perform HTTPS requests and measure the response time and throughput.

[45] The time required to perform these cryprographic operations could potentially be reduced by using hardware crypto-processors.
[46] Cf. section 7.4.2 for a description of this process.
[47] See http://jakarta.apache.org/jmeter/.

While JMeter also supports SOAP requests, it could not be used for measuring the performance of the VSP.

A meaningful performance evaluation of the VSP can only be obtained if a complete TNC check is performed and measured. As described in section 7.3.1, WS-RM is used to establish a session between client and VSP during a TNC check. JMeter, however, does not support WS-RM. Integrating support for WS-RM would require changing JMeter's SOAP module and setting up a complex build process. A more lightweight solution was found with the open source tool soapUI[48]. Although soapUI also does not support WS-RM, it supports the execution of *Groovy*[49] scripts during load tests. Using this extension mechanism, a script was developed for this project that simulates the behaviour of a WS-RM enabled client. In order to perform a load test with soapUI, it is necessary to capture a message exchange that is played back during a load test.

The results of the load tests, which are discussed in the following, were collected in 1100 test runs for each test scenario. In order to minimise the effects of caching or thread ramp-up-time, the first 100 test runs were discarded.

Load Testing the Service Provider In this test, the response times for two Web pages of the SP have been tested: firstly, the Web page that processes the security token issued by the VSP for making an access decision (*TNCCheckResult*), and secondly the Web site that contains the TNC check request (*StartTNCTest*). While the first Web page is generated dynamically while a client is waiting for a reply, the second Web site is almost static[50] and can be returned to the client without performing expensive dynamic operations. This characteristic of the Web pages is reflected in figure 7.26a[51].

This figure shows the response times for both Web pages using different loads. As expected, generating the *TNCCheckResult* page is computational more expensive then returning a static Web page, resulting in a higher response time. The response time for the *TNCCheckResult* page increases nearly linear in average. When doubling the amount of users, the request time grows at the factor 2.01. When doubling the number of clients requesting the static *StartTNCTest* page, however, the request time only increases by the factor 1.35. The difference in the factor and therefore the difference in the curve's slope is caused by the different tasks that are performed when delivering each Web page. As described in section 7.5.2.3, most of the tasks performed while generating the *TNCCheckResult* page are computational expensive, such as parsing XML and performing cryptographic operations. As a consequence, situations in which calculations in one thread can be performed while waiting for I/O activities in other threads occur rarely.

[48] See http://sourceforge.net/projects/soapui/.
[49] Groovy is a dynamic, Java-like scripting language. See http://groovy.codehaus.org/.
[50] That is, only a navigation bar is generated.
[51] Table 7.8 contains the associated measurement values.

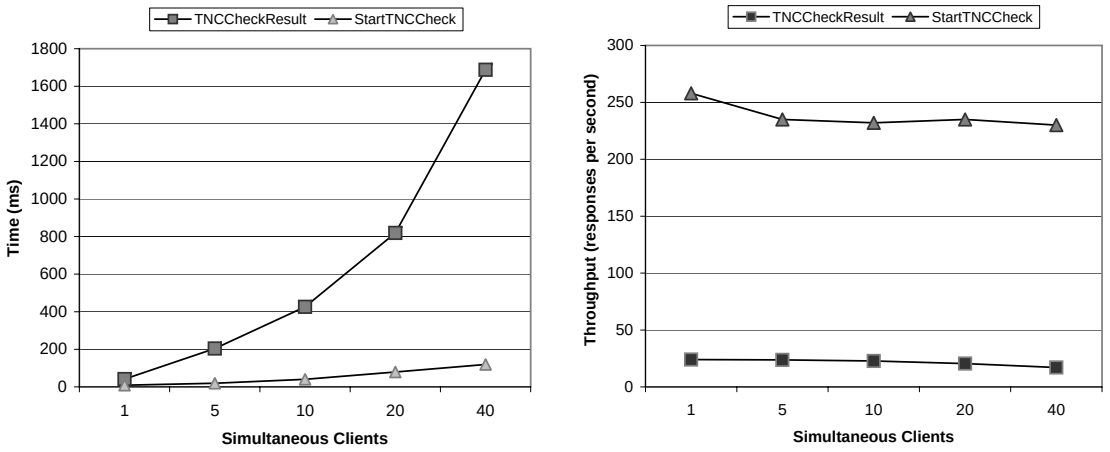


Figure 7.26: Response time (a) and throughput (b) for two Web pages in different load scenarios

Starting with approximately 10 concurrent users, the workload of the CPU reaches 100% when requesting the *TNCCheckResult* page. This can be seen in figure 7.26, which depicts the throughput, that is, the number of replies that the SP can generate per second. At this point the throughput declines visibly (from 22.8 to 17.1 R/s when serving 40 users). Not only is the throughput for retrieving the *StartTNCTest* page notably higher (between 230 and 258 R/s), its relative decline in throughput from serving 10 users to serving 40 is also smaller. There are two reasons why this behaviour was measured. Firstly, the CPU workload of the SP remains below 100% when serving 40 users. The SP is therefore only limited by I/O related factors, such as network or hard drive access times. Secondly, while performing the load tests, the CPU workload of the request generating client reached 100%. The client could therefore not create requests fast enough to exhaust the SP. By adding further clients to the test bed it would be possible to get results that are closer to the maximum throughput. However, because this experiment aimed at showing general load characteristics of an SP rather than the maximum throughput of an IIS when serving almost static content, it was not further investigated.

Table 7.8: Respond times (in ms) and throughput (in responses per second) of the SP during the load test

	TNCCheckResult		**StartTNCCheck**	
Clients	**Throughput (R/s)**	**Response (ms)**	**Throughput (R/s)**	**Response (ms)**
1	24	41	258	9
5	23.7	204	235	19
10	22.8	426	232	40
20	20.5	819	235	79
40	17.1	1688	230	119

Load Testing the Verification Service Provider As mentioned earlier, a test run in the VSP test consists of several requests that are sent sequentially, that is, each simulated user performs a complete TNC check. The results of this approach are depicted in figure 7.27. The response times for each TNC message have been visualised separately. For clarity, the measurement values for all WS-RM message have been combined. Table 7.9 contains the non-aggregated measurement data. The test is based on a TNC check scenario in which three measurements are performed and three WS-Trust messages are sent from a VSP to a client.

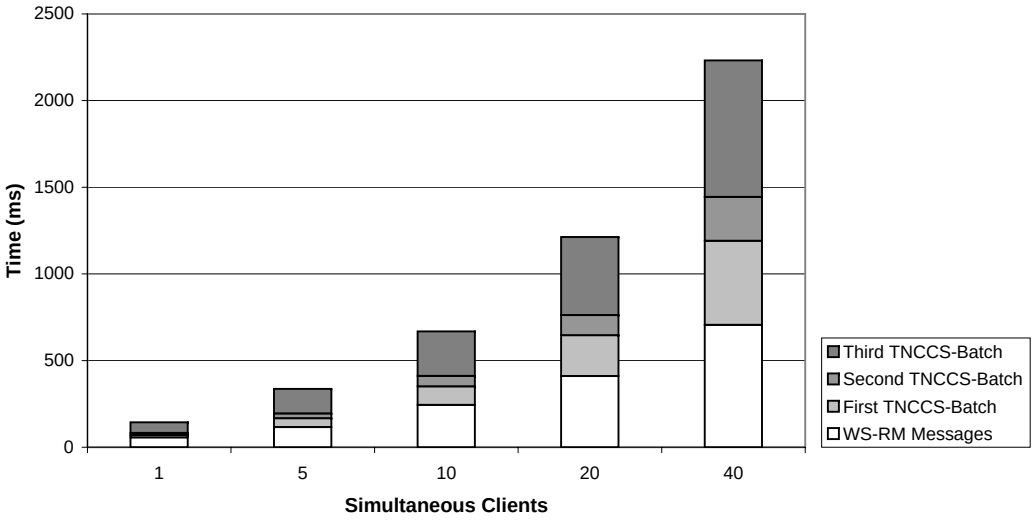


Figure 7.27: Response time for the VSP in different load scenarios

The results obtained in this test are consistent with the results presented in section 7.5.2.2. Comparing a measurement with the integrity policy is a non-expensive task. The time it takes a VSP to send a respond is largely dependant on other tasks. That is, for example, processing VSP and integrity policies before the first TNCCS-Batch and issuing a SAML assertion before the third TNCCS-Batch can be sent to the client. The second TNCCS-Batch, for which no additional tasks have to be performed, is returned in a fraction of the time that is required for the other TNC messages. Furthermore, the third message shows the longest response time. As discussed in section 7.5.2.2, this is mainly related to the costs of issuing a SAML assertion.

As described in section 7.3.1, WS-RM requires that additional messages are exchanged during a TNC check. In average, all WS-RM messages are responsible for about a third (between 31 – 39%) of the total request time during a TNC check.

As described in section 7.3.1, the VSP implementation uses a single *ServerWSStack* object that maps WS-RM session IDs to TNC connections and stores related policy information. For every user, that is, for every new session ID, a new thread is launched. The data held by the *ServerWSStack* object is shared amongst all threads. To prevent data corruption caused

by multiple threads accessing this data simultaneously, a *mutex*[52] is used that synchronises the access to this shared data. Consequently, only one thread at a time can access this data which prevents data from being corrupted. However, this approach also slows down the processing time as threads must wait for other threads to finish accessing the shared data. The data collected in this test reflects this behaviour. When doubling the load for the first message, which initiates a new TNC check and therefore makes excessive use of the *ServerWSStack* object, the response time increases by the factor 2.3. In comparison, the third message is less affected by the mutex and only increases by the factor 1.8. However, when taking all messages into account, the response time increases by the factor 1.9 when doubling the number of users. That is, the VSP implementation can cope with the load in this test and scales linearly.

Table 7.9: Respond times (in ms) of the wTNC VSP during load test in milliseconds

User	WS-RM-1	1. Batch	2. Batch	3. Batch	WS-RM3	WS-RM4	Total
1	29.34	13.94	11.58	61.81	11.4	15.65	143.72
5	48.08	51.12	27.8	141.73	42.8	25.3	383.98
10	141	107.13	60.46	257.0	60.7	42.27	668.56
20	225.67	235.5	116.12	450.54	109.36	75.6	1212.79
40	428.64	485.23	252.8	787.85	163.87	113.3	2231.69

7.6 Summary and Conclusion

The ideas proposed in the previous chapters are based on existing standards. It was therefore possible to leverage existing libraries for parts of the implemented prototype instead of implementing all functionality from scratch. However, the integration of these libraries proved to be challenging. The techniques that have been used to realise the web-based TNC check range from Javascript (in the browser extension) to C# and C. As described in this chapter, solutions have been found to integrate these different components in a way that promotes lose coupling and tight cohesion[53].

The assertion-based architecture describes three entities for enabling a web-based TNC check. The interfaces IF-T and IF-PAA describe how these entities can communicate. The implementation of these entities and their interfaces has been described in this chapter.

A Web browser extension was implemented that enables a web-based TNC check in Firefox Web browsers. It launches the wTNC client that has been implemented for performing TNC checks. In addition, this application manages and combines the user's VSP policy with the VSP policy of an SP and indicates intersections using the developed icon-based system.

[52] See, for example, [TW06, page 81].

[53] See http://martinfowler.com/ieeeSoftware/coupling.pdf for a discussion about coupling and cohesion in software design.

The wTNC client interacts with the wTNC VSP implementation, which has been implemented as a console application, during a TNC check. The communication is based on WS-Trust messages and makes use of WS-Trust extensions. However, these extensions are not part of commonly available frameworks and had to be implemented.

Both the VSP and the client application rely on libTNC for integral TNC tasks, such as the creation of TNCCS-Batches. However, care has been taken to allow the exchange of the underlying TNC library without changing other parts of the applications. Furthermore, the helper application WMIReporter has been developed, which can be used to gather security related integrity measurements.

SPs can state their integrity requirements using a policy. In order to allow multiple SPs to share one VSP, it is necessary that each TNC check can be associated with its integrity policy. Before this policy can be used, it needs to be translated into an IMV specific language. A mechanism was implemented that performs this translation between the high level integrity policy as stated by an SP and an IMV implementation that is part of libTNC. This mechanism has been integrated in compliance with the TNC specifications, that is, other IMVs and TNC libraries can use the same mechanism.

An advantage of the assertion-based architecture is that it imposes only minor changes for SPs that wish to implement a web-based TNC check. To demonstrate this, an example SP has been implemented. It uses the TNC check result to decide whether to grant a client access to a protected area.

In order to evaluate the implemented components, they have been combined in a test bed approach. While a TNC check is not a performance critical process, it is important that it can be performed in a reasonable time without affecting the authentication process. In addition to functional tests, the test bed was therefore used to evaluate the performance of the implemented solution.

The results show that using the test bed a web-based TNC check can be performed in reasonable time (less than 2 seconds in the single user scenario). The total time of a TNC test depends on the type and number of measurements that are performed. Executing simple measurements, such as gathering operating system parameters, is a non expensive task. The required time can increase if other methods are used for gathering measurements, such as those that are used in the WMIReporter. In general, however, the execution time of measurements is exceeded by the time it takes to marshal integrity measurements and access recommendations. Existing libraries have been leveraged and extended in order to provide much of the marshalling functionality. Because these libraries are currently not in a final release state, it is expected that their performance will improve in future releases.

However, the time it takes to complete a TNC check naturally depends on the number of messages that are exchanged. In order to minimise this time, it is therefore important to minimise the number of exchanged WS-Trust RST/RSTR messages. In general, one WS-Trust message can contain several measurements or measurement requests. As described in

section 7.3.2, the integrity policy controls how many messages need to be exchanged during a TNC check. By defining this policy carefully, the total number of exchanged messages can be controlled.

The results of the performed load tests show the behaviour of SP and VSP in situations with more than one concurrent user. The performance characteristics of the SP depend on the type of Web page that is requested. Delivering a Web page that validates a security token requires more processing time. This reduces the throughput and increases the response time compared to delivering almost static Web pages. The VSP load test revealed a performance bottleneck caused by multiple threads accessing shared data. However, the effects of this bottleneck are limited because only a subset of all messages is affected. Overall, the VSP implementation scales with an increasing number of requests and response times increase linearly.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This thesis proposes a mechanism for realising TNC in web-based environments. Current approaches for requesting a security state before granting access to a client rely on users to manually check the security state of their PCs. The previous chapters propose a system based on TNC that allows service providers to automate this process. Instead of relying on users, the security state of a client is measured and compared to an integrity policy. The result of this comparison can be used by service providers to make an access decision.

The original TNC architecture, as described in its specification, is not defined for web-based environments. If it were applied to this environment without modifications, several problem areas arise. The privacy of users would be threatened because TNC integrity measurements reveal details about their PCs to service providers. This exposure of information would also have security implications. A malicious service provider could potentially obtain information about security vulnerabilities of the client and use this information to derive attack strategies. Additionally, when using the original TNC architecture, service providers would need to understand and assess integrity measurement which would result in additional overhead.

This thesis proposes an assertion-based attestation architecture, which addresses the above issues. This architecture makes use of components of the original TNC architecture. However, it changes the flow of information during a TNC check and thereby minimises the impact of a TNC check for the service provider as well as decreasing the risk of privacy and security violations. Instead of performing a TNC check with every service provider, a user performs it with a trusted VSP. Only the result of the TNC check, that is, an access recommendation, is disclosed to service providers. This result can then be used during a service provider's access decision process. Policies are used to express integrity requirements and to state which VSPs are considered trustworthy by SPs.

In order to integrate the TNC check into existing authentication systems, it is important to encapsulate the TNC check result. A mechanism for encapsulating a TNC check result using

SAML assertions has been proposed in this thesis. This approach reduces the complexity of performing a TNC check for a service provider to reading a SAML assertion. SAML also provides integrity protection and, as an open standard, is widely supported and libraries that support SAML assertions are available.

The protocols that are defined for using TNC in network environments are not applicable to web-based environments. This thesis therefore proposes a protocol stack that allows TNC integrity measurements to be exchanged in web-based environments. The focus while designing this stack was on using open standards rather than creating new protocols or protocol extensions. The proposed protocol stack is based on SOAP messages and uses the message formats defined in WS-Trust. WS-Trust's ability to request security tokens, such as SAML assertions, has been leveraged for TNC. In the proposed mechanism, the security token issue process depends on the integrity measurements and integrity requirements that are expressed in a policy. The returned security token reflects the TNC check result.

Before integrity measurements can be exchanged, a service provider needs to request a TNC check. As the TNC check needs to be triggered by a web-application, the TNC check request must be intercepted by a Web browser. This mechanism is restrained to the limited functionality that a Web browser offers. The proposed approach for triggering a TNC check is therefore based on standards that are supported by off-the-shelf Web browsers. The TNC check request is encapsulated in an XHTML page. A custom XML-based format for expressing a TNC check request has been defined. A browser plug-in detects this format and initiates the TNC check request. A service provider can state its integrity and VSP policy as part of the TNC check request.

The system proposed in this thesis caters for both open and closed web-based environments. Using the VSP policy, the trustworthiness of VSPs can be expressed implicitly, that is, based on their capabilities which are assessed by a third party. This option is flexible and allows users to freely choose a VSP in open and closed environments. To simplify VSP policies in closed environments, service providers can state trustworthy VSPs explicitly, that is, by using URLs that point to, for example, a company's VSP. Furthermore, two modes for performing an authentication with a VSP have been considered. In open environments users perform an authentication directly with the VSP. In closed environments a single-sign-on mechanism can be applied that allows a user to reuse the authentication with the SP for authenticating with the VSP.

While previous work on applying TNC in other areas exists, this thesis presents the first implementation of TNC in an area that is not covered by the TNC specification. The proposed mechanisms have been combined in a prototype implementation that consists of a browser extension, a client application for performing the TNC check, a verification service provider that makes an access recommendation, and a service provider that evaluates this recommendation. This implementation was assembled in a test bed approach and successfully demonstrated that a TNC check in web-based environments can be performed.

Performance tests have been conducted that indicate that the TNC check can be performed during an authentication within a reasonable time.

This thesis has shown how TNC can be used in a web-based environment. The proposed solution addresses both open and closed environments. In order to adapt TNC, architectural changes and protocol replacements based on open standards have been proposed. While these changes were necessary, the main concepts of TNC, such as TNCCS-Batches, have been retained. This provides for compatibility with existing TNC integrity measurement software.

8.2 Future Work

The prototype that was implemented for this project can be used to perform a web-based TNC check. However, areas of possible improvement exist, such as performance or user interface optimisations. Furthermore, the addition of two features would greatly enhance a web-based TNC check. Both features have not been implemented in the prototype implementation, mainly because they are only rudimentarily described in the TNC specification.

As described in section 2.4, TNC can be combined with Trusted Platform Modules (TPMs). This combination enhances the trust that a service provider has in the TNC check result, as TPMs are realised in hardware that cannot be affected by malicious software. For example, TPMs can ensure that only genuine Integrity Measurements Collectors are loaded. However, the TNC specification have not yet defined the interface IF-PTS that is used to communicate between TNC and a TPM. Once this interface is defined, TPM support can be added to web-based TNC.

The TNC specification introduces the concept of remediation. If a client does not comply to an integrity policy, it can receive remediation instructions from the TNC Server. These instructions can for example state that the anti-virus definition database needs to be updated. After a client has followed all remediation instructions, the TNC check can be restarted. This remediation capability is a major advantage of TNC when compared to other integrity check solutions, such as TCG compliant (binary) [Tru07a], property-based [SS04], or semantic [HCF04] remote attestation. However, the mechanics of the remediation process and the role of a user is not defined in the TNC specification. Before a web-based remediation mechanism can be designed, the TNC specification needs to address remediation in greater detail.

The features described above will improve web-based TNC checks. The foundation for TNC integrity checks in web-based environments, however, has been laid in this thesis using open standards.

Appendix A

Installation Instructions

The following sections describe how the prototype which has been implemented for this project can be installed. All applications that have been developed are available in source and binary form on the CD which can be found attached to this thesis.

A.1 Client

The client-part of the implemented solution consists of three components. It has been tested under Windows XP SP2 and SP3.

A.1.1 wTNC Browser Extension

The wTNC browser extension has been tested with Mozilla Firefox (versions 2.0.x and 3.0.x). The complete source code of this extension can be found in the folder *Client\wTNC browser extension\source*. A binary version is available in the folder *Client\wTNC browser extension\install* in form of an XPI file. To install the extension, simply drag&drop the file *wTNCPlugin.xpi* into an opened Firefox window. Firefox requires the user to confirm the installation, as depicted in figure A.1. After restarting Firefox, the wTNC browser extension is active and listens for TNC check requests. Before TNC checks can be performed, it is necessary to adjust the path to the wTNC client application (Tools -> Add-ons -> wTNC -> Options). Figure 7.2 on page 108 shows a screenshot of the configuration window.

A.1.2 wTNC Client

The source code of the wTNC client is available in the folder *Client\wTNC client\source*. In order to install the wTNC client, copy the contents of the folder *Client\wTNC client\install* to a local folder. As mentioned in the previous section, it is further required to adjust the path setting in the wTNC browser extension so that it points to the file *wTNCClient.exe*.

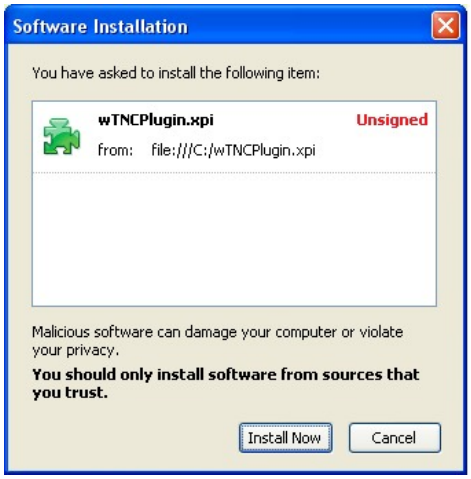


Figure A.1: Installing the wTNC browser extension

A.1.3 WMIReporter

The WMIReporter only needs to be installed if anti-virus or firewall related measurements shall be performed. In this case, the OSC-IMV expects the file *WMIReporter.exe*, which is located in the folder *Client\WMIReporter\install*, to be copied to *C:*. The source code of this application is located in the folder *Client\WMIReporter\source*.

A.2 Service Provider (SP)

The implementation of the SP, that is, the web site *ACME secure Web* has been tested using the Internet Information Server (IIS) version 5.1 and ASP.net version 2.0.5x. The files that are necessary for setting up the SP are located in the folder *SP\source\acmesecureweb*. These files need to be copied to a local folder that can be accessed by the IIS. In the next step, a new virtual directory needs to be set up using the IIS management console (cf. figure A.2). This virtual directory must support the execution of ASP.net files.

In order to enable SSL for the newly created virtual directory, an SSL certificate needs to be obtained (self-signed[1] or from a CA) and assigned to the *Default Web Site* using the IIS management console.

In order to change the VSP and integrity policy, the file *IntegrityPolicyContainer.cs* needs to be edited. This file contains the methods *GetIntegrityPolicyText* and *GetVSPPolicyTextAltSyntax* which are responsible for returning the integrity and VSP policy, respectively.

[1]See section A.3 for instructions on creating a self-signed SSL certificate.

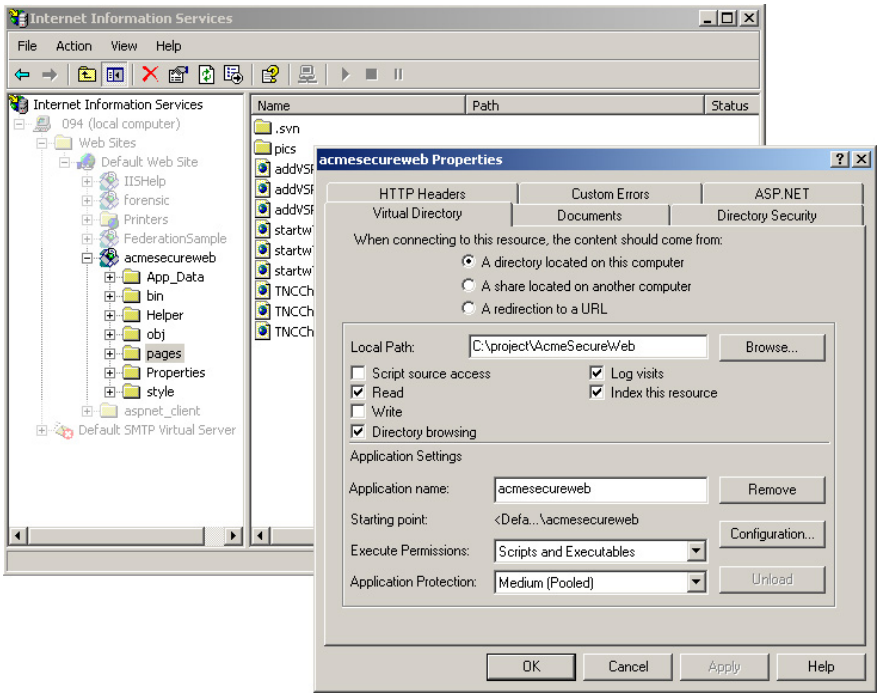


Figure A.2: Configuring IIS 5.1

A.3 Verification Service Provider (VSP)

The VSP software has been tested on a Windows XP SP 2 system with the .Net Framework 3.5. The software is contained in both binary (*VSP\install*) and source (*VSP\source*) form. The main executable file is called *consoleService.exe*. Before the VSP can accept connections, some configuration settings are necessary. The instructions for setting up a VSP are given below.

- Copy the folder *VSP\install* from the CD to a local folder.
- Create the certificates that are used for signing the SAML assertion that contains the TNC check result. Under Windows XP, this can be done by using the command line tool *makecert*[2]. The signing certificate needs to be signed by a certificate that is known and trusted by the SP. If such a root certificate does not exist, it can be created and copied to the SP's certificate store. The following command creates a self signed root certificate and stores it in the certificate store.

  makecert -sv devwtncAroot.pvk -cy authority -r devwtncAroot.cer -a sha1 -n "CN=Dev wTNC Certification Authority" -ss my -sr localmachine

  Based on this root certificate, a signing certificate can be created using the following command:

[2]This tool can be downloaded from http://msdn.microsoft.com/en-us/netframework/aa731542.aspx. See http://msdn.microsoft.com/en-us/library/bfsktky3(VS.80).aspx for a description of all arguments.

makecert.exe -pe -iv devwtncAroot.pvk -ic devwtncAroot.cer -cy end -a sha1 -n CN="VSP A" -sr LocalMachine -ss My -sky exchange -sk VSPA

This certificate is issued for *VSP A* and is stored in the certificate store.

- In Windows XP, the certificate store can be accessed using the *Microsoft Management Console (MMC)* by adding the certificate snap-in. Figure A.3 shows a screenshot of the MMC. The certificates created by the commands above will be placed into the *personal* certificate store on the local computer. In order to install the certificates, they must be moved to an appropriate store location. The root certificate must be placed into the *Trusted Root Certification Authorities* store of the VSP and all SPs that trust this certificate. In order to transfer the certificate to an SP, it can be exported into a file. The signing certificate must be moved into the *Trusted People* store so that it can be found by the VSP.

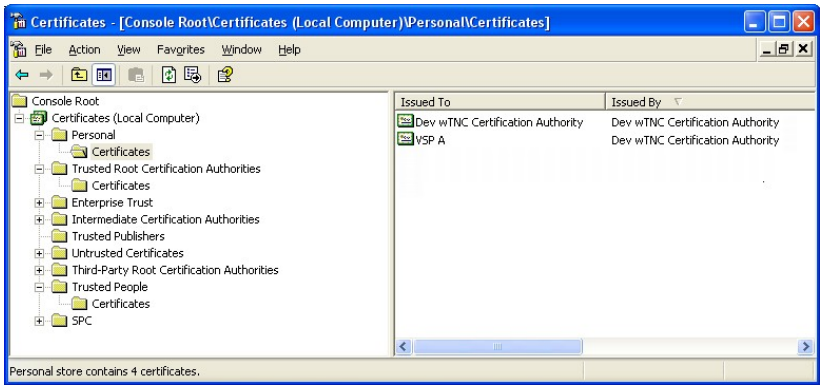


Figure A.3: A screenshot of the certificate snap-in in the MMC

The communication between client and the VSP is protected using TLS. The following description assumes that no existing SSL/TLS certificate can be used for securing this communication. If a certificate exists, the creation of new certificates can be skipped.

- Before a certificate that can be used for protecting the TLS channel can be created, a self-signed root certificate needs to be created. The following command creates such a root certificate.

  makecert -sv devsslroot.pvk -cy authority -r devsslroott.cer -a sha1 -n "CN=Dev SSL Certification Authority" -ss my -sr localmachine

- Based on the root certificate, the following command creates a certificate that can be used for protecting the communication between client and VSP. *HOSTNAME* must be replaced with the domain name or IP address of the VSP.

  makecert -iv devsslroot.pvk -ic devsslroott.cer -cy end -pe -n CN="HOSTNAME" -eku 1.3.6.1.5.5.7.3.1 -ss my -sr localmachine -sky exchange -sp "Microsoft RSA SChannel Cryptographic Provider" -sy 12

- Before the certificate can be used, the root certificate must be moved to the *Trusted Root Certification Authorities* store. This step also needs to be performed on all clients.
- The VSP is a *self hosted application*, that is, it does not rely on a Web server for providing a web-based service. Instead, the VSP relies on the operating system for providing the TLS channel. The command line tool *httpcfg* can be used to configure the TLS channel[3]. The command *httpcfg query ssl* lists all existing TLS connections. An existing TLS connection can be removed using the command *httpcfg delete ssl -i ip:port*. In order to add a new TLS connection, the following command can be used.

  httpcfg set ssl -i ip:port -h CertificateThumbPrint

  CertificateThumbPrint must be replaced by the thumbprint value of the certificate that shall be used for this connection. This value can be obtained from the certificate property dialogue[4].

Finally, the VSP configuration file *consoleService.exe.config* must be adapted so that it reflects the system's configuration. Listing A.1 shows the relevant section. In line 3, the path to the configuration file that contains the list of all IMV modules that shall be loaded by the TNCS must be set. Line 6 contains the issuer name of the certificate that is used to sign SAML assertions.

```
1  <wTNCConfig>
2    <TNCSConfigPath>
3      <add key="path" value="CD\VSP\install\IMVs\tnc_config" />
4    </TNCSConfigPath>
5    <SupportedSigningCA>
6      <add key="CN" value="CN=Dev wTNC Certification Authority" />
7    </SupportedSigningCA>
8  </wTNCConfig>
```

Listing A.1: A section of the VSP configuration file

[3]This application can be downloaded from http://www.microsoft.com/downloads/details.aspx?FamilyID=49ae8576-9bb9-4126-9761-ba8011fabf38&displaylang=en.

[4]Note that any blank spaces must be removed from the thumbprint value before it can be used for httpcfg.

Appendix B

Class Diagrams

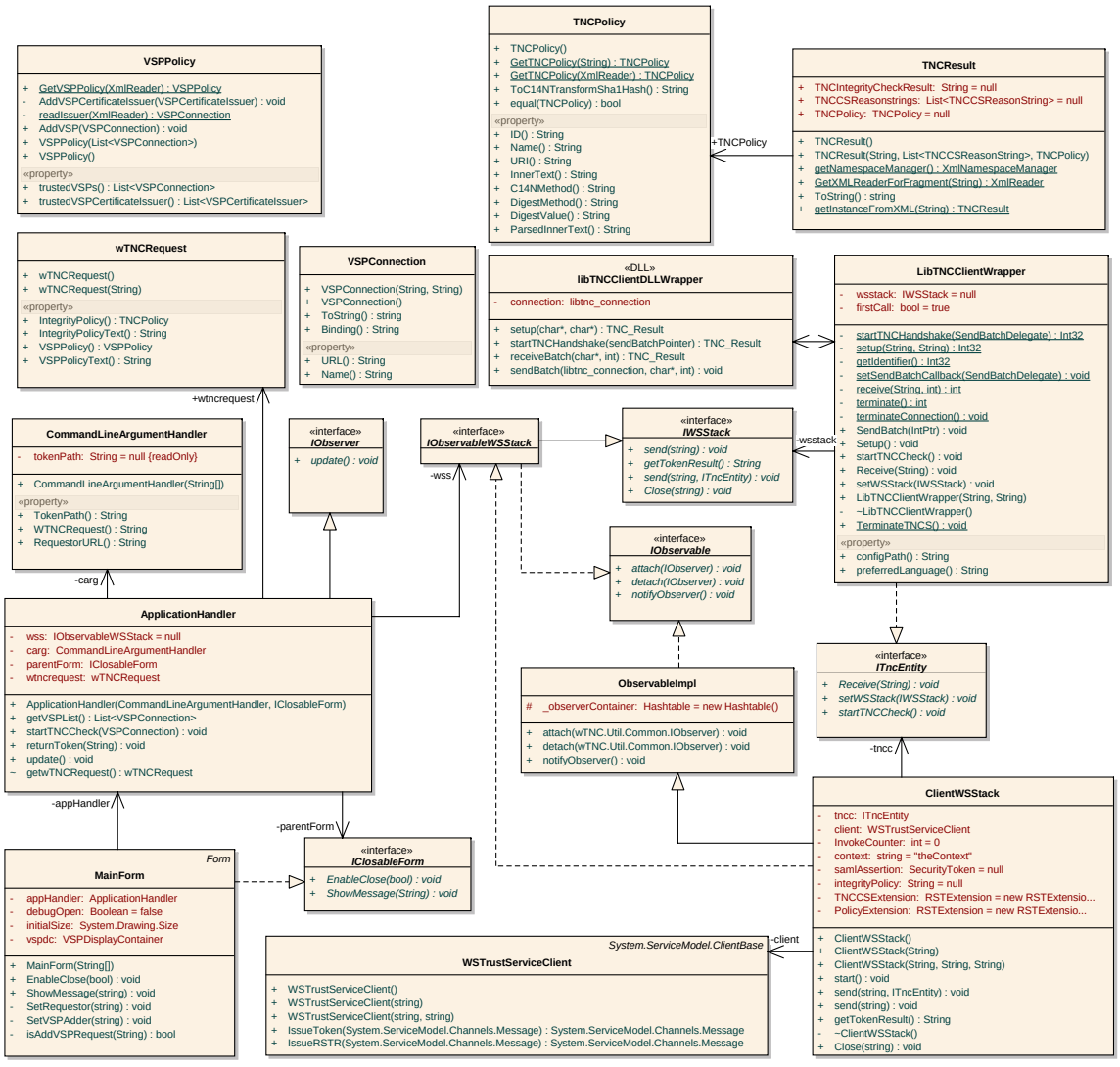


Figure B.1: Class Diagram showing aspects of the wTNC client

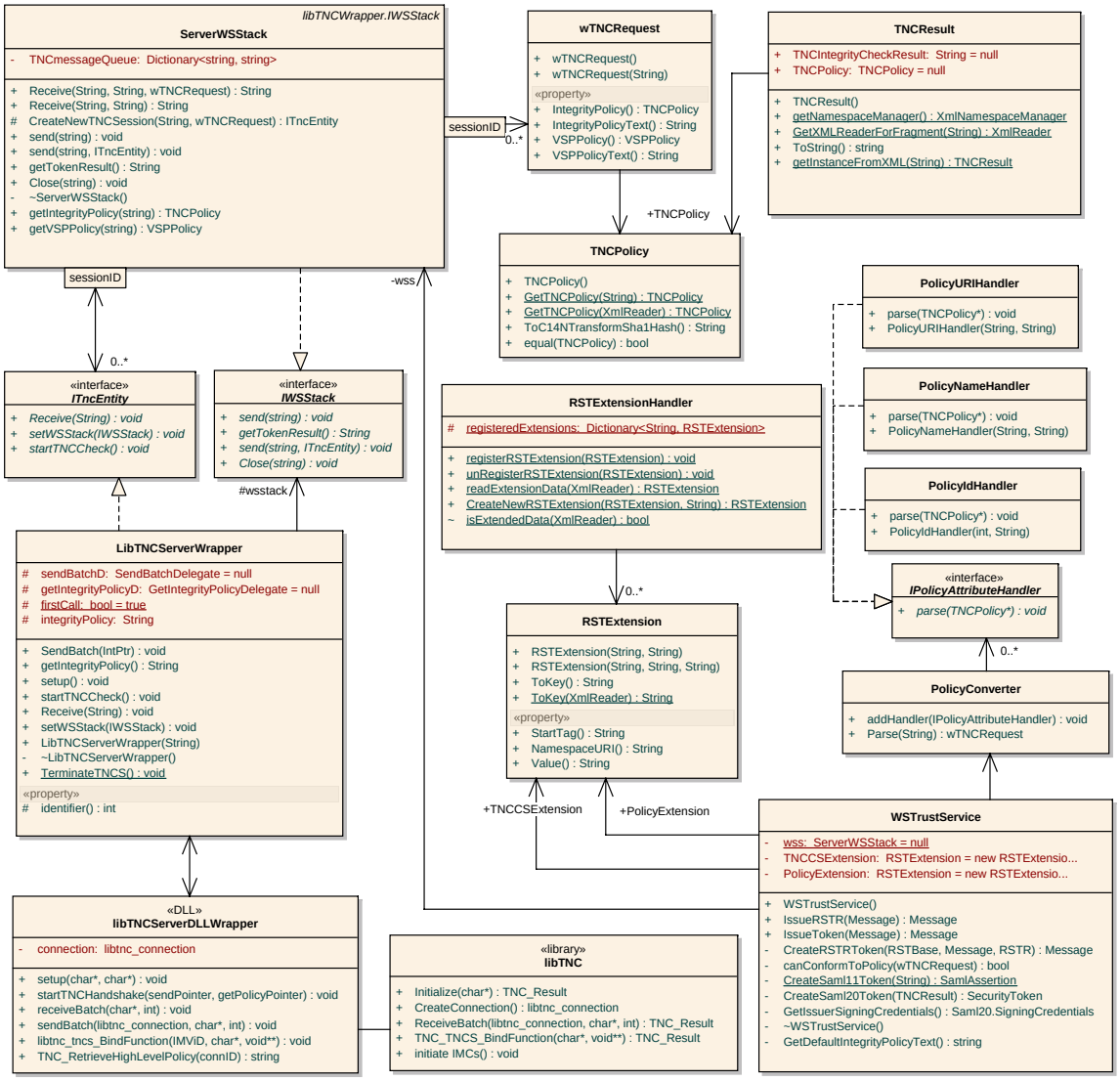


Figure B.2: Class diagram showing aspects of the wTNC VSP

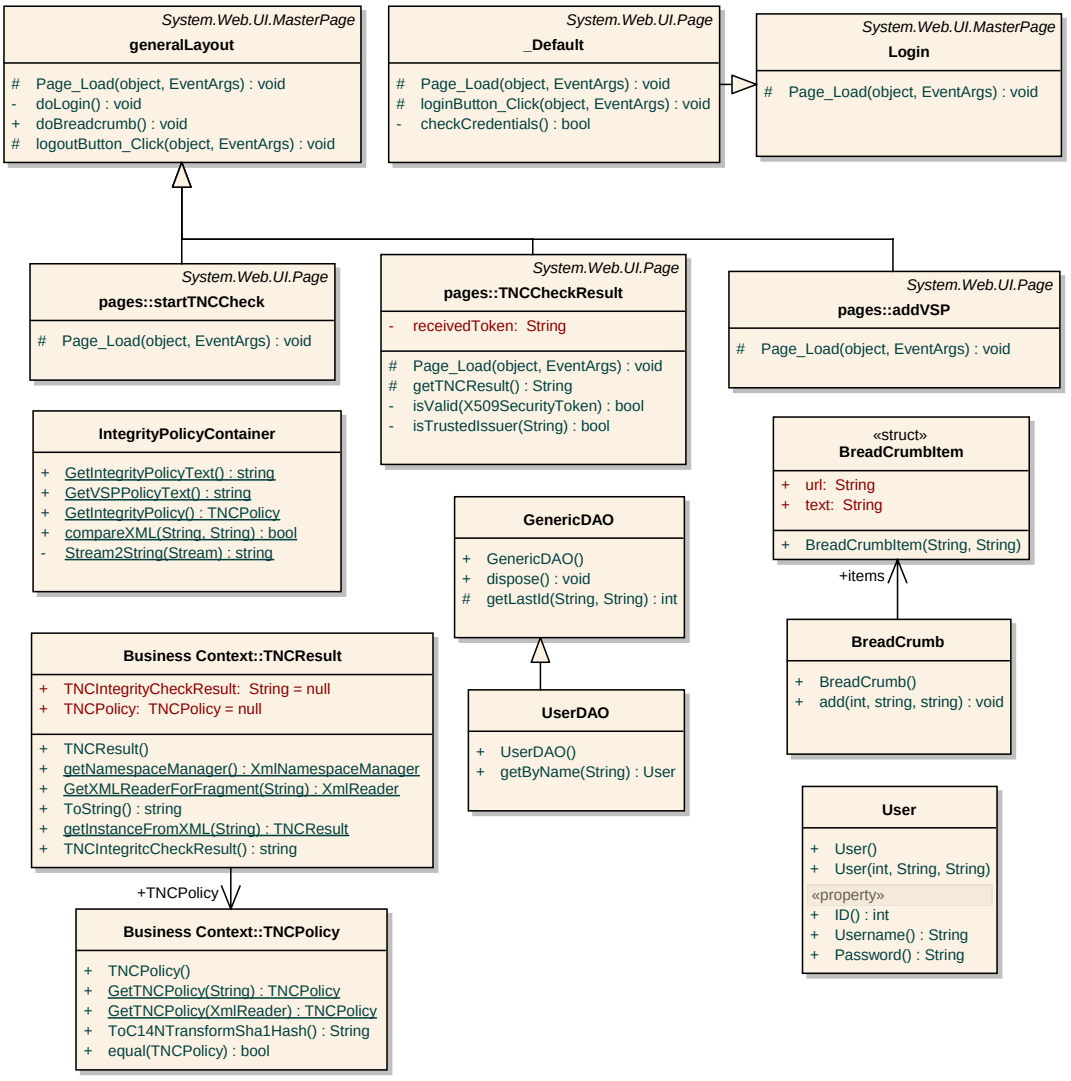


Figure B.3: Class diagram showing aspects of the SP implementation

Appendix C

Message Dialogue

The following listings show the complete message exchange between VSP and client during a TNC check, that is, from requesting a TNC check to receiving the result of the test.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:a="http://www.w3.org/2005/08/addressing"
    xmlns:u="http://docs.oasis-open.org/wss/2004
    /01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
 <s:Header>
  <a:Action s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequence
      </a:Action>
  <a:MessageID>urn:uuid:76cb80df-0f91-4a08-aa50-7e0ff047be50</a:MessageID>
  <a:To s:mustUnderstand="1">https://cosc852.cosc.canterbury.ac.nz:8012/TNCService</a:To>
  <o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-open.org/wss/2004
      /01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <u:Timestamp u:Id="_0">
    <u:Created>2008-11-24T21:43:44.578Z</u:Created>
    <u:Expires>2008-11-24T21:48:44.578Z</u:Expires>
   </u:Timestamp>
   <o:UsernameToken u:Id="uuid-777b53e4-c6bd-4218-b3a8-976036efb3af-1">
    <o:Username>
     <!-- Removed-->
    </o:Username>
    <o:Password>
     <!-- Removed-->
    </o:Password>
   </o:UsernameToken>
  </o:Security>
 </s:Header>
 <s:Body>
  <CreateSequence xmlns="http://schemas.xmlsoap.org/ws/2005/02/rm">
   <AcksTo>
    <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
   </AcksTo>
   <Offer>
    <Identifier>urn:uuid:2bbb13ef-3236-496b-b9a4-5849e1f589cd</Identifier>
   </Offer>
  </CreateSequence>
 </s:Body>
</s:Envelope>
```

Listing C.1: Establishing a new session with WS-RM. This message is sent from the client to the VSP.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:a="http://www.w3.org/2005/08/addressing"
    xmlns:u="http://docs.oasis-open.org/wss/2004/
    01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
 <s:Header>
  <a:Action
      s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequenceResponse
      </a:Action>
  <a:RelatesTo>urn:uuid:76cb80df-0f91-4a08-aa50-7e0ff047be50</a:RelatesTo>
  <o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-open.org/wss/2004/
      01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <u:Timestamp u:Id="_0">
    <u:Created>2008-11-24T21:43:50.612Z</u:Created>
    <u:Expires>2008-11-24T21:48:50.612Z</u:Expires>
   </u:Timestamp>
  </o:Security>
 </s:Header>
 <s:Body>
  <CreateSequenceResponse xmlns="http://schemas.xmlsoap.org/ws/2005/02/rm">
   <Identifier>urn:uuid:9a7666a2-501a-4ca7-accf-7599192dd10c</Identifier>
   <Accept>
    <AcksTo>
     <a:Address>https://cosc852.cosc.canterbury.ac.nz:8012/TNCService</a:Address>
    </AcksTo>
   </Accept>
  </CreateSequenceResponse>
 </s:Body>
</s:Envelope>
```

Listing C.2: Establishing a new session with WS-RM. This message is sent from the VSP to the client.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
    xmlns:a="http://www.w3.org/2005/08/addressing"
    xmlns:u="http://docs.oasis-open.org/wss/2004/
    01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
 <s:Header>
  <r:Sequence s:mustUnderstand="1">
   <r:Identifier>urn:uuid:9a7666a2-501a-4ca7-accf-7599192dd10c</r:Identifier>
   <r:MessageNumber>1</r:MessageNumber>
  </r:Sequence>
  <a:Action
      s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue</a:Action>
  <a:MessageID>urn:uuid:c53b7acf-5041-4475-917e-e0bad84a90ea</a:MessageID>
  <a:ReplyTo>
   <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
  </a:ReplyTo>
  <a:To s:mustUnderstand="1">https://cosc852.cosc.canterbury.ac.nz:8012/TNCService</a:To>
  <o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-open.org/wss/2004/
      01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <u:Timestamp u:Id="_0">
    <u:Created>2008-11-24T21:43:46.234Z</u:Created>
    <u:Expires>2008-11-24T21:48:46.234Z</u:Expires>
   </u:Timestamp>
   <o:UsernameToken u:Id="uuid-777b53e4-c6bd-4218-b3a8-976036efb3af-1">
    <o:Username>
     <!-- Removed-->
    </o:Username>
```

```
    <o:Password>
     <!-- Removed-->
    </o:Password>
   </o:UsernameToken>
  </o:Security>
 </s:Header>
 <s:Body>
  <RequestSecurityToken xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">
   <TokenType>urn:oasis:names:tc:SAML:1.0:assertion</TokenType>
   <RequestType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue</RequestType>
   <t:TNCCS-Batch BatchId="1" Recipient="TNCS"
       xmlns:t="http://www.trustedcomputinggroup.org/IWG/TNC/1_0/IF_TNCCS#"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemalocation="http://www.trustedcomputinggroup.org/IWG/TNC/1_0/IF_TNCCS#
       https://www.trustedcomputinggroup.org/XML/SCHEMA/TNCCS_1.0.xsd">
    <t:TNCC-TNCS-Message>
     <t:Type>00000003</t:Type>
     <t:XML>
      <t:TNCCS-PreferredLanguage/>
     </t:XML>
    </t:TNCC-TNCS-Message>
    <t:IMC-IMV-Message>
     <t:Type>00235801</t:Type>
     <t:Base64>V2luZG93c3w1fDF8MjYwMHwyfFNlcnZpY2UgUGFjayAzfDN8MHwwMzAwfDE=</t:Base64>
    </t:IMC-IMV-Message>
   </t:TNCCS-Batch>
   <wTNCPolicy xmlns="http://www.canterbury.ac.nz/wTNC">
    <wtncp:integritypolicy xmlns:wtncp="http://www.canterbury.ac.nz/wtip">
     <wsp:policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
      <wsp:exactlyone>
       <wsp:all>
        <wtncp:windowsoperatingsystem>
         <wsp:exactlyone>
          <wsp:all>
           <wtncp:personalfirewall>
            <wtncp:active/>
           </wtncp:personalfirewall>
          </wsp:all>
         </wsp:exactlyone>
        </wtncp:windowsoperatingsystem>
       </wsp:all>
      </wsp:exactlyone>
     </wsp:policy>
    </wtncp:integritypolicy>
    <wtvp:vsppolicy xmlns:wtnc="http://www.canterbury.ac.nz/wtvp">
     <wsp:policy xmlns:wsp="http://www.w3.org/ns/ws-policy">
      <wsp:exactlyone>
       <wsp:all>
        <sp:issuedtoken
            sp:includetoken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/
            IncludeToken/AlwaysToRecipient"
            xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
         <wsp:exacltyone>
          <wsp:exacltyone>
           <sp:issuer xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
            <wsa:address>https://cosc852.cosc.canterbury.ac.nz:8012/TNCService</wsa:address>
```

```
            <wtvp:vspname>Secure Issuer</wtnc:vspname>
           </sp:issuer>
           <sp:issuer>
            <wsa:address>https://127.0.0.1:8012/TNCService</wsa:address>
            <wtvp:vspname>Certificate Error</wtnc:vspname>
           </sp:issuer>
           <sp:issuer>
            <wsa:address>http://localhost:8010/TNCService</wsa:address>
            <wtvp:vspname>Unsecure Issuer</wtnc:vspname>
           </sp:issuer>
          </wsp:exacltyone>
          <sp:recipientsignaturetoken>
           <sp:x509token
               sp:includetoken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/
               IncludeToken/AlwaysToRecipient">
            <wsp:policy>
             <wsp:exacltyone>
              <sp:issuername>CN=Dev wTNC Certification Authority A</sp:issuername>
             </wsp:exacltyone>
            </wsp:policy>
           </sp:x509token>
          </sp:recipientsignaturetoken>
         </wsp:exacltyone>
        </sp:issuedtoken>
       </wsp:all>
      </wsp:exactlyone>
     </wsp:policy>
    </wtvp:vsppolicy>
   </wTNCPolicy>
  </RequestSecurityToken>
 </s:Body>
</s:Envelope>
```

Listing C.3: Initial RST message sent from the client to the VSP. This message contains the first TNCCS-Batch, integrity policy, and VSP policy.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
    xmlns:a="http://www.w3.org/2005/08/addressing"
    xmlns:u="http://docs.oasis-open.org/wss/2004/
    01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
 <s:Header>
  <r:Sequence s:mustUnderstand="1">
   <r:Identifier>urn:uuid:2bbb13ef-3236-496b-b9a4-5849e1f589cd</r:Identifier>
   <r:MessageNumber>1</r:MessageNumber>
  </r:Sequence>
  <r:SequenceAcknowledgement>
   <r:Identifier>urn:uuid:9a7666a2-501a-4ca7-accf-7599192dd10c</r:Identifier>
   <r:AcknowledgementRange Lower="1" Upper="1"/>
  </r:SequenceAcknowledgement>
  <a:Action s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue
      </a:Action>
  <a:RelatesTo>urn:uuid:c53b7acf-5041-4475-917e-e0bad84a90ea</a:RelatesTo>
  <o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-open.org/wss/2004/
      01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <u:Timestamp u:Id="_0">
```

```
    <u:Created>2008-11-24T21:43:50.643Z</u:Created>
    <u:Expires>2008-11-24T21:48:50.643Z</u:Expires>
   </u:Timestamp>
  </o:Security>
 </s:Header>
 <s:Body>
  <RequestSecurityTokenResponse xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">
   <t:TNCCS-Batch BatchId="2" Recipient="TNCC"
        xmlns:t="http://www.trustedcomputinggroup.org/IWG/TNC/1_0/IF_TNCCS#"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemalocation="http://www.trustedcomputinggroup.org/IWG/TNC/1_0/IF_TNCCS#
        https://www.trustedcomputinggroup.org/XML/SCHEMA/TNCCS_1.0.xsd">
     <t:IMC-IMV-Message>
       <t:Type>00235809</t:Type>
       <t:Base64>RklSRVdBTEwgQ0hFQ0tfQU5Z</t:Base64>
      </t:IMC-IMV-Message>
   </t:TNCCS-Batch>
  </RequestSecurityTokenResponse>
 </s:Body>
</s:Envelope>
```

Listing C.4: This RSTR message is received by the client from the VSP and contains a TNCCS-Batch that requests additional measurements.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
     xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
     xmlns:a="http://www.w3.org/2005/08/addressing"
     xmlns:u="http://docs.oasis-open.org/wss/2004/
     01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
 <s:Header>
  <r:SequenceAcknowledgement>
   <r:Identifier>urn:uuid:2bbb13ef-3236-496b-b9a4-5849e1f589cd</r:Identifier>
   <r:AcknowledgementRange Lower="1" Upper="1"/>
  </r:SequenceAcknowledgement>
  <r:Sequence s:mustUnderstand="1">
   <r:Identifier>urn:uuid:9a7666a2-501a-4ca7-accf-7599192dd10c</r:Identifier>
   <r:MessageNumber>2</r:MessageNumber>
  </r:Sequence>
  <a:Action
       s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue</a:Action>
  <a:MessageID>urn:uuid:7a353153-09bf-4df5-ad72-2a21563a3e92</a:MessageID>
  <a:ReplyTo>
   <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
  </a:ReplyTo>
  <a:To s:mustUnderstand="1">https://cosc852.cosc.canterbury.ac.nz:8012/TNCService</a:To>
  <o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-open.org/wss/2004/
      01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <u:Timestamp u:Id="_0">
    <u:Created>2008-11-24T21:43:46.421Z</u:Created>
    <u:Expires>2008-11-24T21:48:46.421Z</u:Expires>
   </u:Timestamp>
   <o:UsernameToken u:Id="uuid-777b53e4-c6bd-4218-b3a8-976036efb3af-1">
    <o:Username>
     <!-- Removed -->
    </o:Username>
    <o:Password>
```

```
      <!-- Removed-->
     </o:Password>
    </o:UsernameToken>
   </o:Security>
  </s:Header>
  <s:Body>
   <RequestSecurityTokenResponse xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">
    <TokenType>urn:oasis:names:tc:SAML:1.0:assertion</TokenType>
    <t:TNCCS-Batch BatchId="3" Recipient="TNCS"
         xmlns:t="http://www.trustedcomputinggroup.org/IWG/TNC/1_0/IF_TNCCS#"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemalocation="http://www.trustedcomputinggroup.org/IWG/TNC/1_0/IF_TNCCS#
         https://www.trustedcomputinggroup.org/XML/SCHEMA/TNCCS_1.0.xsd">
     <t:IMC-IMV-Message>
      <t:Type>0023580a</t:Type>
      <t:Base64>RklSRVdBTEwgQ0hFQ0tfQU5ZfDF8MWAX</t:Base64>
     </t:IMC-IMV-Message>
    </t:TNCCS-Batch>
   </RequestSecurityTokenResponse>
  </s:Body>
</s:Envelope>
```

Listing C.5: RSTR message contains requested measurements. This message is sent from the client to the VSP.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
     xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
     xmlns:a="http://www.w3.org/2005/08/addressing"
     xmlns:u="http://docs.oasis-open.org/wss/2004/
     01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
 <s:Header>
  <r:Sequence s:mustUnderstand="1">
   <r:Identifier>urn:uuid:2bbb13ef-3236-496b-b9a4-5849e1f589cd</r:Identifier>
   <r:MessageNumber>2</r:MessageNumber>
  </r:Sequence>
  <r:SequenceAcknowledgement>
   <r:Identifier>urn:uuid:9a7666a2-501a-4ca7-accf-7599192dd10c</r:Identifier>
   <r:AcknowledgementRange Lower="1" Upper="2"/>

  </r:SequenceAcknowledgement>
  <a:Action
       s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue</a:Action>
  <a:RelatesTo>urn:uuid:7a353153-09bf-4df5-ad72-2a21563a3e92</a:RelatesTo>
  <o:Security s:mustUnderstand="1" xmlns:o="http://docs.oasis-open.org/wss/2004/
      01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <u:Timestamp u:Id="_0">
    <u:Created>2008-11-24T21:43:50.846Z</u:Created>
    <u:Expires>2008-11-24T21:48:50.846Z</u:Expires>
   </u:Timestamp>
  </o:Security>
 </s:Header>
 <s:Body>
  <RequestSecurityTokenResponse xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">
   <TokenType>urn:oasis:names:tc:SAML:1.0:assertion</TokenType>
   <t:TNCCS-Batch BatchId="4" Recipient="TNCC"
        xmlns:t="http://www.trustedcomputinggroup.org/IWG/TNC/1_0/IF_TNCCS#"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
      xsi:schemalocation="http://www.trustedcomputinggroup.org/IWG/TNC/1_0/IF_TNCCS#
      https://www.trustedcomputinggroup.org/XML/SCHEMA/TNCCS_1.0.xsd">
    <t:TNCC–TNCS–Message>
      <t:Type>00000001</t:Type>
      <t:XML>
        <t:TNCCS–Recommendation type="allow"></t:TNCCS–Recommendation>
      </t:XML>
    </t:TNCC–TNCS–Message>
  </t:TNCCS–Batch>
  <RequestedSecurityToken>
   <Assertion Version="SAML 2.0" ID="SAML_acf2946d-e241-489c-87dc-d224c113f0bc"
       IssueInstant="2008-11-24T21:43:50.846Z"
       xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
    <Issuer>cosc852</Issuer>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
     <SignedInfo>
      <CanonicalizationMethod
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="#SAML_acf2946d-e241-489c-87dc-d224c113f0bc">
       <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
       </Transforms>
       <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <DigestValue>5LN5Hl6wPGGmBgraSmDZGGmXCtU=</DigestValue>
      </Reference>
     </SignedInfo>
     <SignatureValue>F6HDiwvaWR3AhV1YfpEfEtTQ7Bw...stMwXpz/c=</SignatureValue>
     <KeyInfo>
      <X509Data>
       <X509Certificate>MIICKjCCAZegAwIBAgIQs...hr0fLsFcIRcZ</X509Certificate>
      </X509Data>
     </KeyInfo>
    </Signature>
    <Subject>
     <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
      <SubjectConfirmationData NotBefore="2008-11-25T10:43:50.846Z"
          NotOnOrAfter="2008-11-25T10:53:50.846Z" Address="132.181.12.144"/>
     </SubjectConfirmation>
    </Subject>
    <Conditions NotBefore="2008-11-24T21:43:50.846Z"
        NotOnOrAfter="2008-11-24T21:53:50.846Z">
     <OneTimeUse/>
    </Conditions>
    <AttributeStatement>
     <Attribute Name="TNCResult" NameFormat="wTNCFormat">
      <TNCResult xmlns="http://www.canterbury.ac.nz/wTNCResult"
          xmlns:x="http://www.w3.org/2000/09/xmldsig#>
       <TNCIntegrityCheckResult>compliant</TNCIntegrityCheckResult>
       <wTNCPolicy>
        <x:Transforms>
         <x:Transform x:Algorithm="http://www.w3.org/2006/12/xml–c14n11"/>
        </x:Transforms>
        <x:DigestMethod x:Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <x:DigestValue>Qx4SSuQCPuIFgyMJych1er/pyhs=</x:DigestValue>
```

```
        </wTNCPolicy>
       </TNCResult>
      </Attribute>
     </AttributeStatement>
    </Assertion>
   </RequestedSecurityToken>
  </RequestSecurityTokenResponse>
 </s:Body>
</s:Envelope>
```

Listing C.6: RSTR message sent from the VSP to the client. It contains the issued SAML assertion and the TNCS Action Recommendation.

Appendix D

XML Schemas

In the following, XML Schema representations of the formats that have been proposed in this thesis are given.

D.1 TNC Result

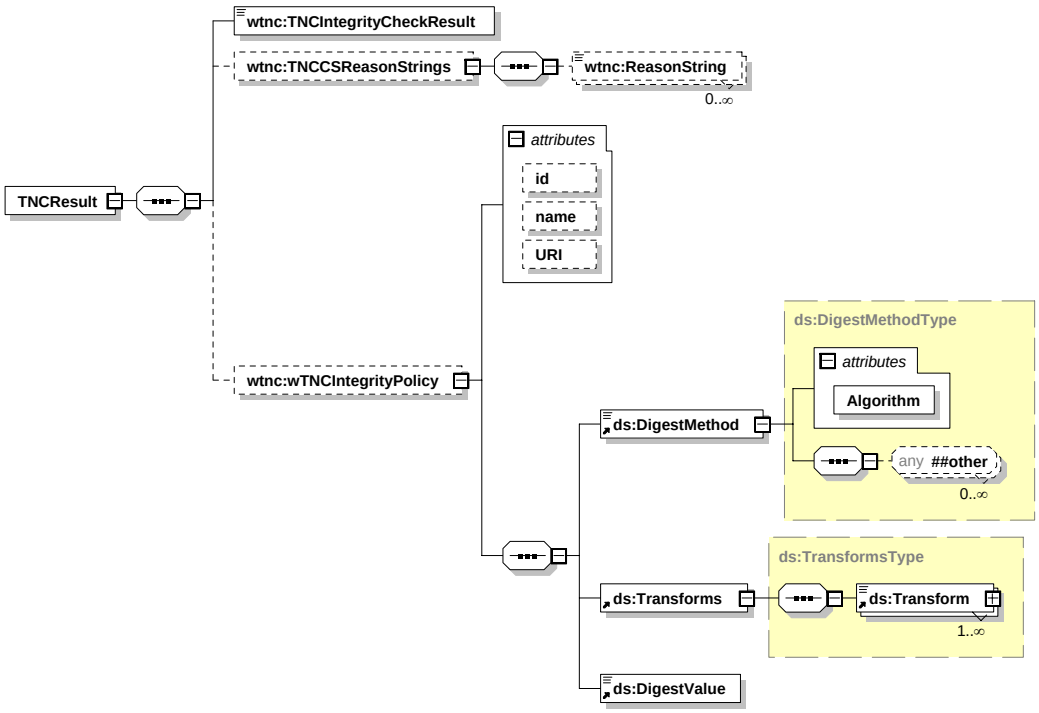


Figure D.1: Graphical representation of the XML Schema of the TNC result

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns="http://www.canterbury.ac.nz/wTNCResult"
    xmlns:wtnc="http://www.canterbury.ac.nz/wTNCResult"
```

```
    targetNamespace="http://www.canterbury.ac.nz/wTNCResult"
    elementFormDefault="qualified" attributeFormDefault="unqualified" version="0.1">
 <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
     schemaLocation="http://www.w3.org/2000/09/xmldsig#"/>
 <xs:simpleType name="TNCIntegrityCheckResult">
  <xs:restriction base="xs:string">
   <xs:enumeration value="compliant"/>
   <xs:enumeration value="non-compliant"/>
   <xs:enumeration value="indeterminate"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="TNCResult">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="TNCIntegrityCheckResult" type="wtnc:TNCIntegrityCheckResult"/>
    <xs:element name="TNCCSReasonStrings" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="ReasonString" type="xs:string" minOccurs="0"
           maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="wTNCIntegrityPolicy" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="ds:DigestMethod"/>
       <xs:element ref="ds:Transforms"/>
       <xs:element ref="ds:DigestValue"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string"/>
      <xs:attribute name="name" type="xs:string"/>
      <xs:attribute name="URI" type="xs:anyURI"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Listing D.1: XML Schema of the TNC result

D.2 Integrity Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns="http://www.canterbury.ac.nz/wtip" xmlns:wtnc="http://www.canterbury.ac.nz/wtip"
    targetNamespace="http://www.canterbury.ac.nz/wtip" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
 <xs:import namespace="http://schemas.xmlsoap.org/ws/2004/09/policy"
     schemaLocation="http://schemas.xmlsoap.org/ws/2004/09/policy"/>
 <xs:complexType name="WindowsOperatingSystem">
```

```
 <xs:sequence>
  <xs:element ref="wsp:ExactlyOne" minOccurs="0"/>
  <xs:element name="Version" minOccurs="0"/>
  <xs:element name="ServicePack" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="PersonalFirewall">
 <xs:sequence>
  <xs:element ref="wsp:ExactlyOne" minOccurs="0"/>
  <xs:element name="Vendor" type="ApplicationDetails" minOccurs="0"/>
  <xs:element name="Product" minOccurs="0">
   <xs:complexType>
    <xs:simpleContent>
     <xs:extension base="ApplicationDetails">
     <xs:sequence>
       <xs:element name="MinProductVersion" type="xs:integer"/>
      </xs:sequence>
     </xs:extension>
    </xs:simpleContent>
   </xs:complexType>
  </xs:element>
  <xs:element name="Active" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="ApplicationDetails">
 <xs:simpleContent>
  <xs:extension base="xs:string">
   <xs:attribute name="name" type="xs:string"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="Antivirus">
 <xs:sequence>
  <xs:element ref="wsp:ExactlyOne" minOccurs="0"/>
  <xs:element name="Vendor" type="ApplicationDetails" minOccurs="0"/>
  <xs:element name="Product" minOccurs="0">
   <xs:complexType>
    <xs:simpleContent>
     <xs:extension base="ApplicationDetails">
     <xs:sequence>
       <xs:element name="MinProductVersion" type="xs:integer"/>
      </xs:sequence>
     </xs:extension>
    </xs:simpleContent>
   </xs:complexType>
  </xs:element>
  <xs:element name="UpToDate" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
<xs:element name="IntegrityPolicy">
 <xs:annotation>
  <xs:documentation/>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="wsp:Policy"/>
```

```
    </xs:sequence>
    <xs:attribute name="id" type="xs:string"/>
    <xs:attribute name="name" type="xs:string"/>
    <xs:attribute name="URI" type="xs:anyURI"/>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

Listing D.2: XML Schema of the integrity policy

D.3 VSP Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:wtnc="http://www.canterbury.ac.nz/wTNC"
     targetNamespace="http://www.canterbury.ac.nz/wTNC" elementFormDefault="qualified"
     attributeFormDefault="unqualified">
 <xs:import namespace="http://schemas.xmlsoap.org/ws/2004/09/policy"
      schemaLocation="http://schemas.xmlsoap.org/ws/2004/09/policy"/>
 <xs:element name="VSPPolicy">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="wsp:Policy"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="VSPName" type="xs:string"/>
</xs:schema>
```

Listing D.3: XML Schema of the VSP policy

D.4 TNC Check Request

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:wtvp="http://www.canterbury.ac.nz/wtvp"
     xmlns:wtip="http://www.canterbury.ac.nz/wtip"
     xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust" elementFormDefault="qualified"
     attributeFormDefault="unqualified">
 <xs:import namespace="http://www.canterbury.ac.nz/wtvp"
      schemaLocation="http://www.canterbury.ac.nz/wtvp"/>
 <xs:import namespace="http://www.canterbury.ac.nz/wtip"
      schemaLocation="http://www.canterbury.ac.nz/wtip"/>
 <xs:import namespace="http://schemas.xmlsoap.org/ws/2005/02/trust"
      schemaLocation="http://schemas.xmlsoap.org/ws/2005/02/trust/WS-Trust.xsd"/>
 <xs:element name="TNCIntegrityCheckRequest">
  <xs:complexType>
   <xs:sequence>
```

```
    <xs:element ref="wtvp:VSPPolicy"/>
    <xs:element ref="wtip:IntegrityPolicy"/>
    <xs:element name="SecurityToken">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="wst:RequestedSecurityToken"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Listing D.4: XML Schema of the TNC check request

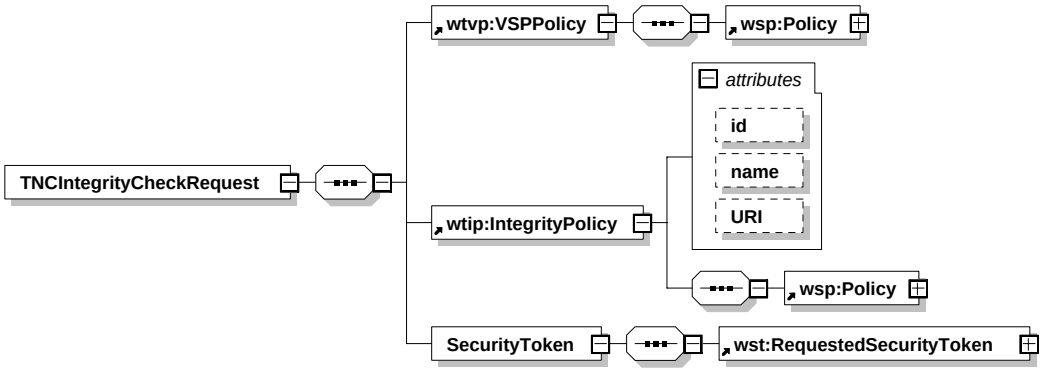


Figure D.2: Graphical representation of the XML Schema of the TNC check request

List of Figures

List of Tables

Listings

Bibliography

[Abo04] ABOBA ET AL: *RFC 3748 - Extensible Authentication Protocol (EAP)*. `http://www.ietf.org/rfc/rfc3748.txt`. Version: June 2004

[AC00] ABOBA, B. ; CALHOUN, P.: *RFC 3579 - RADIUS Support For Extensible Authentication Protocol*. `http://www.ietf.org/rfc/rfc2869.txt`. Version: June 2000

[AM07a] ALRODHAN, Waleed A. ; MITCHELL, Chris J.: Addressing privacy issues in CardSpace. In: *IAS 2007: Third International Symposium on Information Assurance and Security* (2007), pages 285–291

[AM07b] ALTHEIM, Murray ; MCCARRON, Shane: *XHTML 1.1 - Module-based XHTML - Second Edition - W3C Working Draft*. `http://www.w3.org/TR/2007/WD-xhtml11-20070216/`. Version: February 2007

[Ant08] ANTI-PHISHING WORKING GROUP: *Phishing Activity Trends Report Q1 2008*. `http://www.antiphishing.org/reports/apwg_report_Q1_2008.pdf`. Version: 2008

[APM+06] AUSTIN, Daniel ; PERUVEMBA, Subramanian ; MCCARRON, Shane ; ISHIKAWA, Masayasu ; BIRBECK, Mark: *XHTM Modularization 1.1 - W3C Working Draft*. `http://www.w3.org/TR/2006/WD-xhtml-modularization-20060705/`. Version: July 2006

[AVKK+04] AU, Richard ; VASANTA, Harikrishna ; KIM-KWANG ; CHOO, Raymond ; LOOI, Mark: A user-centric anonymous authorisation framework in e-commerce environment. In: *ICEC '04: Proceedings of the 6th international conference on Electronic commerce*. New York, NY, USA : ACM, 2004, pages 138–147

[Bak03] In: BAKKEN, David E.: *Middleware*. Kluwer Academic Press

[BB04] BRUHN, Russel ; BURTON, Philip: Displaying mathematics in a web browser using MathML and SVG. In: *MSCCC '04: Proceedings of the 2nd annual conference on Mid-south college computing*. Little Rock, Arkansas, United States, 2004, pages 97–106

[BBF+02] BARTEL, Mark ; BOYER, John ; FOX, Barb ; LAMACCHIA, Brian ; SIMON, Ed ; EASTLAKE, Donald (editor) ; REAGLE, Joseph (editor) ; DAVIDSOLO (editor):

XML-Signature Syntax and Processing - W3C Recommendation. http://www.w3.org/TR/xmldsig-core/. Version: February 2002

[BDK+07] BOYER, John M. ; DUBINKO, Micah ; KLOTZ, Leigh L. ; LANDWEHR, David ; MERRICK, Roland ; RAMAN, T. V.: *XForms 1.0 (Third Edition) W3C Recommendation*. http://www.w3.org/TR/2007/REC-xforms-20071029/. Version: October 2007

[BHK+03] BOX, Don ; HONDO, Maryann ; KALER, Chris ; MARUYAMA, Hiroshi ; NAGARATNAM, Nataraj ; PATRICK, Paul ; RIEGEN, Claus von ; SHEWCHUK, John ; NADALIN, Anthony (editor): *Web Services Policy Assertions Language (WS-PolicyAssertions)*. http://xml.coverpages.org/ws-policyassertionsV11.pdf. Version: May 2003

[BKB00] BOUCH, Anna ; KUCHINSKY, Allan ; BHATTI, Nina: Quality is in the eye of the beholder: meeting users' requirements for Internet quality of service. In: *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 2000, pages 297–304

[Bla06] BLAKE-WILSON ET AL: *RFC 4366 - Transport Layer Security (TLS) Extensions*. http://www.ietf.org/rfc/rfc4366.txt. Version: April 2006

[BLMT04] BENJUMEA, Vicente ; LOPEZ, Javier ; MONTENEGRO, Jose A. ; TROYA, Jose M.: A First Approach to Provide Anonymity in Attribute Certificates. In: *PKC 2004: Public Key Cryptography* (2004), pages 402–415

[BM07] BALFE, Shane ; MOHAMMED, Anish: Final Fantasy - Securing On-Line Gaming with Trusted Computing. In: *Autonomic and Trusted Computing* (2007), pages 123–134

[Bra06] BRAY, Tim (editor) ; PAOLI, Jean (editor) ; SPERBERG-MCQUEEN, C. M. (editor) ; MALER, Eve (editor) ; YERGEAU, François (editor)- World Wide Web Consortium (W3C): *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. http://www.w3.org/TR/xml/. Retrieved 04.03.2008

[BSB08] BERTOCCI, Vittorio ; SERACK, Garrett ; BAKER, Caleb: *Understanding Windows CardSpace: An Introduction to the Concepts and Challenges of Digital Identities (Independent Technology Guides)*. Addison-Wesley Professional, 2008. ISBN 0321496841

[CHJ07] CHOI, SuGil ; HAN, JinHee ; JUN, SungIk: Improvement on TCG Attestation and Its Implication for DRM. In: *ICCSA 2007: Computational Science and Its Applications*, pages 912–925

[CIMP03] CARLISLE, David ; ION, Patrick ; MINER, Robert ; POPPELIER, Nico: *Mathematical Markup Language (MathML) Version 2.0 (Second Edition) - W3C Recom-*

mendation. http://www.w3.org/TR/2003/REC-MathML2-20031021/. Version: October 2003

[CLL+06] CHEN, Liqun ; LANDFERMANN, Rainer ; LÖHR, Hans ; ROHE, Markus ; SADEGHI, Ahmad-Reza ; STÜBLE, Christian: A protocol for property-based attestation. In: *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*. New York, NY, USA : ACM, 2006, pages 7–16

[Com07a] COMPUTER EMERGENCY RESPONSE TEAM (CERT), CARNEGIE MELLON UNIVERSITY: *CERT Statistics*. http://www.cert.org/stats/cert_stats.html. Retrieved 03.04.2008

[Com07b] COMPUTER SECURITY INSTITUTE: *CSI Survey 2007. The twelfth annual computer crime and security survey.* http://www.gocsi.com/forms/csi_survey.jhtml. Version: 2007

[Dav08] DAVIS, Doug (editor) ; KARMARKAR, Anish (editor) ; PILZ, Gilbert (editor) ; WINKLER, Steve (editor) ; YALÇINALP Ümit (editor): *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1*. OASIS Standard, January 2008

[DH76] DIFFIE, Whitfield ; HELLMAN, Martin E.: New Directions in Cryptography. In: *IEEE Transactions on Information Theory* IT-22 (1976), No. 6, pages 644–654

[EA03] EDNEY, Jon ; ARBAUGH, William A.: *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*. Addison-Wesley Professional, 2003. ISBN 0321136209

[ET05] ERONEN, P. ; TSCHOFENIG, H.: *RFC 4279 - Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*. http://www.ietf.org/rfc/rfc4279.txt. Version: 2005

[FH02] FARRELL, S. ; HOUSLEY, R.: *RFC 3281 - An Internet Attribute Certificate Profile for Authorization*. http://www.ietf.org/rfc/rfc3281.txt. Version: April 2002

[For07] FOROUZAN, Behrouz A.: *Cryptography & Network Security*. McGraw-Hill, 2007. ISBN 0073327530

[Fun03] FUNK, Paul: *EAP-TTLS Status*. http://www.ietf.org/proceedings/05mar/slides/eap-7/eap-7.ppt. Retrieved 05.03.2008

[Fur07] FURNELL, Steven: An assessment of website password practices. In: *Computers & Security* 26 (2007), No. 7-8, pages 445–451

[FZML02] FARKAS, Csilla ; ZIEGLER, Gábor ; MERETEI, Attila ; LÖRINCZ, András: Anonymity and accountability in self-organizing electronic communities. In: *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*. New York, NY, USA : ACM, 2002, pages 81–90

[GHJV94] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John M.: *Design Patterns: Elements of Reusable Object-Oriented Software (Addison-Wesley Professional Computing Series)*. Addison-Wesley Professional, 1994. ISBN 9780201633610

[GHM+07] GUDGIN, Martin ; HADLEY, Marc ; MENDELSOHN, Noah ; MOREAU, Jean-Jacques ; NIELSEN, Henrik F. ; KARMARKAR, Anish ; LAFON, Yves: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) - W3C Recommendation*. April 2007

[GHR06] GUDGIN, Martin ; HADLEY, Marc ; ROGERS, Tony: *Web Services Addressing 1.0 - Core – W3C Recommendation*. `http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/`. Version: May 2006

[GM06] GALLERY, Eimear ; MITCHELL, Chris: Trusted Computing Technologies and their use in the Provision of High Assurance SDR Platforms. In: *Proceedings of 2006 Software Defined Radio Technical Conference*

[GPS06] GOLDMAN, Kenneth ; PEREZ, Ronald ; SAILER, Reiner: Linking remote attestation to secure tunnel endpoints. In: *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*. New York, NY, USA : ACM Press, 2006, pages 21–24

[Gre03] GREGORY, Kate: *Microsoft Visual C++ .NET 2003 Kick Start*. Sams, 2003. ISBN 9780672326004

[GSS+07] GASMI, Yacine ; SADEGHI, Ahmad-Reza ; STEWIN, Patrick ; UNGER, Martin ; ASOKAN, N.: Beyond secure channels. In: *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*. ISBN 978–1–59593–888–6, pages 30–40

[Gud03] GUDGIN, Martin (editor) ; HADLEY, Marc (editor) ; MENDELSOHN, Noah (editor) ; MOREAU, Jean-Jacques (editor) ; NIELSEN, Henrik F. (editor) World Wide Web Consortium (W3C): *SOAP Version 1.2 Part 1: Messaging Framework - W3C Recommendation*. June 2003

[Gud07] GUDGIN, Martin (editor) ; HADLEY, Marc (editor) ; MENDELSOHN, Noah (editor) ; MOREAU, Jean-Jacques (editor) ; NIELSEN, Henrik F. (editor) ; KARMARKAR, Anish (editor) ; LAFON, Yves (editor): *SOAP Version 1.2 Part 1: Adjuncts (Second Edition)*. April 2007

[Hal06] HALDAR, Vivek: *Semantic Remote Attestation*, University of California, Phd thesis, 2006. Online-Resource. `http://www.vivekhaldar.com/blog/wp-content/uploads/2006/03/thesis.pdf`

[HCF04] HALDAR, V. ; CHANDRA, D. ; FRANZ, M.: Semantic remote attestation: A

virtual machine directed approach to trusted computing. In: *USENIX Virtual Machine Research and Technology Symposium*, 2004

[HM04] HUGHES, John ; MALER, Eve: *Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1.1*. 2004

[Int05] INTERNATIONAL TELECOMMUNICATION UNION: *ITU-T Recommendation X.509. Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks*. August 2005

[KMW08] KARUSSEIT, Martin ; MARGARIA, Tiziana ; WILLEBRANDT, Holger: Policy expression and checking in XACML, WS-Policies, and the jABC. In: *TAV-WEB '08: Proceedings of the 2008 workshop on Testing, analysis, and verification of web services and applications*. New York, NY, USA : ACM, 2008, pages 20–26

[KN06] KOSEK, Jirka ; NÁLEVKA, Petr: Relaxed: on the way towards true validation of compound documents. In: *WWW '06: Proceedings of the 15th international conference on World Wide Web*. New York, NY, USA : ACM, 2006, pages 427–436

[KSP07] KOLTER, Jan ; SCHILLINGER, Rolf ; PERNUL, Guenther: A Privacy-Enhanced Attribute-Based Access Control System. In: *Data and Applications Security XXI* (2007), pages 129–143

[KSS07] KÜHN, Ulrich ; SELHORST, Marcel ; STÜBLE, Christian: Realizing property-based attestation and sealing with commonly available hard- and software. In: *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*. New York, NY, USA : ACM, 2007, pages 50–57

[LAB+06] LOCKHART, Hal ; ANDERSEN, Steve ; BOHREN, Jeff ; SVERDLOV, Yakov ; HONDO, Maryann ; MARUYAMA, Hiroshi ; NAGARATNAM, Nataraj ; BOUBEZ, Toufic ; MORRISON, Scott ; NANDA, Arun ; SCHMIDT, Don ; WALTERS, Doug ; WILSON, Hervey ; BURCH, Lloyd ; EARL, Doug ; BAJA, Siddharth ; PRAFULLCHANDRA, Hemma ; NADALIN, Anthony (editor) ; KALER, Chris (editor): *Web Services Federation Language (WS-Federation) Version 1.1*. December 2006

[LAN04] LAN/MAN STANDARDS COMMITEE OF THE IEEE COMPUTER SOCIETY: *Standard for Local and Metropolitan Area Networks - Port-Based Network Access Control - IEEE Std. 802.1X*. 2004

[LKS06] LAITKORPI, Markku ; KOSKINEN, Johannes ; SYSTA, Tarja: A UML-based Approach for Abstracting Application Interfaces to REST-like Services. Los Alamitos, CA, USA : IEEE Computer Society, 2006, pages 134–146

[LM01] LAVY, Matthew ; MEGGITT, Ashley: *Windows Management Instrumentation (WMI)*. Sams, 2001. ISBN 9781578702602

[Low07] LOWY, Juval: *Programming WCF Services (Programming)*. O'Reilly Media, Inc., 2007. ISBN 9780596526993

[MB06] M. BROWN, R. H.: *Transport Layer Security (TLS) Authorization Extensions*. http://tools.ietf.org/wg/tls/draft-housley-tls-authz-extns-07.txt. Version: June 2006

[McC08] MCCAULEY, Mike: *libTNC – A TNC implementation for Windows and Unix*. May 2008

[Mic08] MICROSOSFT CORPORATION: *Frequently asked questions about Windows Security Center*. http://support.microsoft.com/kb/883792. Retrieved 02.09.2008

[Mit07] MITRA, Nila (editor) ; LAFON, Yves (editor) World Wide Web Consortium (W3C): *SOAP Version 1.2 Part 0: Primer (Second Edition)*. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/. Version: April 2007

[MMP03] MALER, Eve ; MISHRA, Prateek ; PHILPOTT, Rob: *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1*. http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf. Version: 2003

[Mon06] MONZILLO, Ronald (editor) ; KALER, Chris (editor) ; NADALIN, Anthony (editor) ; HALLEM-BAKER, Phillip (editor): *Web Services Security: SAML Token Profile 1.1 - OASIS Standard*. February 2006

[Nad07] NADALIN, Anthony (editor) ; GOODNER, Marc (editor) ; GUDGIN, Martin (editor) ; BARBIR, Abbie (editor) ; GRANQVIST, Hans (editor): *WS-Trust 1.3 – OASIS Standard*. March 2007

[Nan07] NANDA, Arun: *Identity Selector Interoperability Profile V1.0*. http://download.microsoft.com/download/1/1/a/11ac6505-e4c0-4e05-987c-6f1d31855cd2/Identity-Selector-Interop-Profile-v1.pdf. Version: April 2007

[Net99a] NETWORK WORKING GROUP: *RFC 2246 - The TLS Protocol Version 1.0*. http://www.ietf.org/rfc/rfc2246.txt. Version: January 1999

[Net99b] NETWORK WORKING GROUP: *RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1*. http://www.ietf.org/rfc/rfc2616.txt. Version: 1999

[Net00] NETWORK WORKING GROUP: *RFC 2818 - HTTP Over TLS*. http://www.ietf.org/rfc/rfc2818.txt. Version: May 2000

[Net06a] NETWORK WORKING GROUP: *RFC 4346 - The Transport Layer Security (TLS) Protocol Version 1.1*. http://www.ietf.org/rfc/rfc4346.txt. Version: April 2006

[Net06b] NETWORK WORKING GROUP: *RFC 4366 - Transport Layer Security (TLS) Extensions*. `http://www.ietf.org/rfc/rfc4366.txt`. Version: April 2006

[NGG+07] NADALIN, Anthony ; GOODNER, Marc ; GUDGIN, Martin ; BARBIR, Abbie ; GRANQVIST, Hans: *OASIS Standard – WS-SecurityPolicy 1.2*. July 2007

[Nik02] NIKANDER, Pekka: Denial-of-Service, Address Ownership, and Early Authentication in the IPv6 World. In: *Security Protocols: 9th International Workshop Cambridge, UK, April 25-27, 2001. Revised Papers* (2002), pages 12–21

[NKMHB06] NADALIN, Anthony ; KALER, Chris ; MONZILLO, Ronald ; HALLAM-BAKER, Phillip: *Web Services Security: SOAP Message Security 1.1*. OASIS Standard. `http://docs.oasis-open.org/wss/v1.1/`. Version: February 2006

[NVH07] NAGARAJAN, Aarthi ; VARADHARAJAN, Vijay ; HITCHENS, Michael: Trust management for trusted computing platforms in web services. In: *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*. New York, NY, USA : ACM, 2007, pages 58–62

[OHB06] OPPLIGER, Rolf ; HAUSER, Ralf ; BASIN, David: SSL/TLS session-aware user authentication - Or how to effectively thwart the man-in-the-middle. In: *Computer Communications* 29 (2006), August, No. 12, pages 2238–2246

[OHB+07] OPPLIGER, Rolf ; HAUSER, Ralf ; BASIN, David ; RODENHAEUSER, Aldo ; KAISER, Bruno: A Proof of Concept Implementation of SSL/TLS Session-Aware User Authentication (TLS-SA). In: *Kommunikation in Verteilten Systemen (KiVS)* (2007), pages 225–236

[PA07] *Chapter* Security in Service-Oriented Architecture: Issues, Standards, and Implementations. In: PADMANABHUNI, Srinivas ; ADARKAR, Hemant: *Securing Web Services: Practical Usage of Standards and Specifications*. IGI Global, 2007, pages 1–21

[PDMP05] PRIEBE, Torsten ; DOBMEIER, Wolfgang ; MUSCHALL, Björn ; PERNUL, Günther: ABAC - Ein Referenzmodell für attributbasierte Zugriffskontrolle. In: FEDERRATH, Hannes (editor): *Sicherheit* Bd. 62, GI, 2005, pages 285–296

[PLR07] PENG, Tao ; LECKIE, Christopher ; RAMAMOHANARAO, Kotagiri: Survey of network-based defense mechanisms countering the DoS and DDoS problems. In: *ACM Comput. Surv.* 39 (2007), No. 1, pages 3

[RCB08] RESNICK, Steve ; CRANE, Richard ; BOWEN, Chris: *Essential Windows Communication Foundation (WCF): For .NET Framework 3.5 (Microsoft .NET Development Series)*. Addison-Wesley Professional, 2008. ISBN 9780321440068

[RE04] RANKL, Wolfgang ; EFFING, Wolfgang: *Smart Card Handbook*. 3. Wiley, 2004. ISBN 0471988758

[RHJ99] RAGGETT, Dave ; HORS, Arnaud L. ; JACOBS, Ian: *HTML 4.01 Specification - W3C Recommendation*. http://www.w3.org/TR/html401/. Version: December 1999

[Rig00] RIGNEY ET AL: *RFC 2865 - Remote Authentication Dial In User Service (RADIUS)*. http://www.ietf.org/rfc/rfc2865.txt. Version: June 2000

[RR04] ROSENBERG, Jothy ; REMY, David: *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Sams, 2004. ISBN 0672326515

[SAM03] *Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1*. OASIS Standard, 2003

[SAM05] CANTOR, Scott (editor) ; HIRSCH, Frederick (editor) ; KEMP, John (editor) ; PHILPOTT, Rob (editor) ; MALER, Eve (editor) ; OASIS (Veranst.): *Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0*. http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf. Version: 2005

[SAM08] RAGOUZIS, Nick (editor) ; HUGHES, John (editor) ; PHILPOTT, Rob (editor) ; MALER, Eve (editor) ; MADSEN, Paul (editor) ; SCAVO, Tom (editor): *Security Assertion Markup Language (SAML) V2.0 Technical Overview. Committee Draft 10 - OASIS*. 2008

[SC05] SCOTT CANTOR, Rob Philpott Eve M.: *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf. Version: 2005

[SG07] SCHLAEGER, Christian ; GANSLMAYER, Monika: Effects of Architectural Decisions in Authentication and Authorisation Infrastructures. In: *ARES 2007: The Second International Conference on Availability, Reliability and Security, 2007.* (2007), April, pages 230–237

[SIYL08] SRIVATSA, Mudhakar ; IYENGAR, Arun ; YIN, Jian ; LIU, Ling: Mitigating application-level denial of service attacks on Web servers: A client-transparent approach. In: *ACM Trans. Web* 2 (2008), No. 3, pages 1–49

[SJZD04] SAILER, Reiner ; JAEGER, Trent ; ZHANG, Xiaolan ; DOORN, Leendert van: Attestation-based policy enforcement for remote access. In: *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*. ISBN 1–58113–961–6, pages 308–317

[SL04] SPECHT, Stephen M. ; LEE, Ruby B.: Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures. In: BADER, David A. (ed-

itor) ; KHOKHAR, Ashfaq A. (editor): *International Conference Parallel and Distributed Computer Systems*, ISBN 1–880843–52–8, pages 543–550

[SLM06] SONG, Zhexuan ; LEE, Sung ; MASUOKA, Ryusuke: Trusted Web Service. In: *The Second Workshop on Advances in Trusted Computing*, 2006

[SS75] SALTZER, J.H. ; SCHROEDER, M.D.: The protection of information in computer systems. In: *Proceedings of the IEEE* 63 (1975), Sept., No. 9, pages 1278–1308

[SS04] SADEGHI, Ahmad-Reza ; STÜBLE, Christian: Property-based attestation for computing platforms: caring about properties, not mechanisms. In: *NSPW '04: Proceedings of the 2004 workshop on New security paradigms*. New York, NY, USA : ACM, 2004, pages 67–77

[SSMP06] SCHLAEGER, Christian ; SOJER, Manuel ; MUSCHALL, Bjoern ; PERNUL, Guenther: Attribute-Based Authentication and Authorisation Infrastructures for E-Commerce Providers. In: *E-Commerce and Web Technologies* (2006), pages 132–141

[STRE06] STUMPF, F. ; TAFRESCHI, O. ; RÖDER, P. ; ECKERT, C.: A robust Integrity Reporting Protocol for Remote Attestation. In: *Proceedings of the Second Workshop on Advances in Trusted Computing (WATC 06 Fall)*. Tokyo, Dec 2006

[Sul05] SULLIVAN, Roger K.: The case for federated identity. In: *Network Security* 2005 (2005), September, No. 9, pages 15–19

[Tan03] TANENBAUM, Andrew S.: *Computer Networks*. 4th edition. Prentice Hall, New Jersey, 2003. ISBN 0130661023

[Tim05] TIM MOSES (ED.): *OASIS Standard – eXtensible Access Control Markup Language (XACML) Version 2.0*. February 2005

[TLS06] TLS WORKING GROUP: *TLS Inner Application Extension (TLS/IA)*. http://tools.ietf.org/html/draft-funk-tls-inner-application-extension-03, June 2006

[TLS07] TLS WORKING GROUP: *TLS using EAP Authentication*. http://www.ietf.org/internet-drafts/draft-nir-tls-eap-02.txt, October 2007

[Tru05] TRUSTED COMPUTING GROUP: *Subject Key Attestation Evidence Extension - TCG Specification*. 2005

[Tru07a] TRUSTED COMPUTING GROUP: *TCG Specification Architecture Overview Version 1.4*. 2007

[Tru07b] TRUSTED COMPUTING GROUP: *TCG Trusted Network Connect TNC Architecture for Interoperability - Specification Version 1.2*. 2007

[Tru07c] TRUSTED COMPUTING GROUP: *TCG Trusted Network Connect TNC IF-IMC - TCG Specification*. 2007

[Tru07d] TRUSTED COMPUTING GROUP: *TCG Trusted Network Connect TNC IF-IMV - TCG Specification*. 2007

[Tru07e] TRUSTED COMPUTING GROUP: *TCG Trusted Network Connect TNC IF-PEP: Protocol Bindings for RADIUS - TCG Specification*. 2007

[Tru07f] TRUSTED COMPUTING GROUP: *TCG Trusted Network Connect TNC IF-T: Protocol Binding for Tunneled EAP Methods - TCG Specification*. 2007

[Tru07g] TRUSTED COMPUTING GROUP: *TCG Trusted Network Connect TNC IF-TNCCS - TCG Specification*. 2007

[TS06] TANENBAUM, Andrew S. ; STEEN, Maarten V.: *Distributed Systems: Principles and Paradigms (2nd Edition)*. 2. Prentice Hall, 2006. ISBN 0130888931

[TW06] TANENBAUM, Andrew ; WOODHULL, Albert: *Operating Systems Design and Implementation (3rd Edition) (Prentice Hall Software Series)*. Prentice Hall, 2006. ISBN 9780131429383

[VOH+07] VEDAMUTHU, Asir ; ORCHARD, David ; HIRSCH, Frederick ; HONDO, Maryann ; YENDLURI, Prasad ; BOUBEZ, Toufic ; YALCINALP, Uemit: *W3C Recommendation – Web Services Policy Framework (WS-Policy) 1.5*. 2007

[Wut06] WUTTKE, Daniel: *Erweiterung sicherheitsrelevanter Software für die automatische Integritätsprüfung von Endgeräten*, Fachhochschule Hannover, Germany, Master Thesis, 2006

[YEN+05] YOSHIHAMA, S. ; EBRINGER, T. ; NAKAMURA, M. ; MUNETOH, S. ; MARUYAMA, H.: WS-attestation: efficient and fine-grained remote attestation on Web services. In: *ICWS 2005: IEEE International Conference on Web Services, 2005. Proceedings.*, 2005

[YT05] YUAN, E. ; TONG, J.: Attribut based access control (ABAC) for Web services. In: *ICWS 2005: IEEE International Conference on Web Services. Proceedings* (2005), July

[Zor00] ZORN ET AL: *RFC 2868 - RADIUS Attributes for Tunnel Protocol Support*. `http://www.ietf.org/rfc/rfc2868.txt`. Version: June 2000